ABSTRACT

Title of dissertation: NEUTRAL GAS AND PLASMA INTERACTIONS IN THE POLAR CUSP

David K. Olson, Doctor of Philosophy, 2012

Dissertation directed by: Professor Douglas C. Hamilton
University of Maryland

When the solar wind interacts with the Earth's magnetosphere, both energy and matter can be transferred across the magnetopause boundary. This transfer gives rise to numerous phenomena, including ion outflow and neutral upwelling in the polar cusps. These processes are caused by a transfer of energy to the ionospheric plasma and neutral gas through various mechanisms. The heated plasma or gas expands, increasing the density of the atmosphere at high altitudes by as much as a factor of two, and injecting ionospheric plasma into and even outside of the magnetosphere. These two phenomena are examined in two ways: A novel high energy (0.1-10 keV) spectrograph for ionospheric cusp ions was designed as part of the Rocket Experiment for Neutral Upwelling (RENU), a sounding rocket campaign carried out at the northern polar cusp to observe the electrodynamic properties of the cusp during a neutral upwelling event. This instrument is called the KeV Ion Magnetic Spectrograph (KIMS). Ion outflow in the ionosphere has shown evidence of correlation with both Poynting flux and soft electron precipitation in the cusp. The heat input from these energy sources might also affect neutral gas in the ionosphere, contributing to upwelling phenomena seen at the dayside cusp. Using data from the Fast Auroral Snapshot Explorer (FAST) and the Challenging Minisatellite Payload (CHAMP) satellites, correlations of electromagnetic and particle energy inputs are examined with both ion outflow and neutral upwelling in the cusp. The added ability to process large quantities of data quickly and reference the data between separate satellites in this statistical survey gives clues to the consistency of the observed correlations with ion outflow over time and to the relative importance of these energy sources in the neutral upwelling phenomenon. It also provides the ability to understand these connections in a broad spectrum of conditions of the Sun and solar wind as well as in the Earth's magnetosphere.

NEUTRAL GAS AND PLASMA INTERACTIONS IN THE POLAR CUSP

by

David K. Olson

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2012

Acknowledgments

Though a dissertation is outwardly a very individual experience, no project of this magnitude could ever be undertaken without the assistance of many other people. The time spent in research is supported by many who are even unaware of their help and in ways that I found very surprising in my own experience. I could never express gratitude to all who have supported this project, but I will try to do so for a few particular individuals who have provided unique assistance and opportunity.

My advisors, Dr. Moore and Dr. Coplan, have been very patient in guiding my work and my education in these past years. I appreciate the consistent encouragement and support they offer, and have seen the same extended to other students and colleagues with whom they have worked.

Dr. Marc Lessard, of the University of New Hampshire, provided the opportunity to participate in the RENU sounding rocket campaign. That was a marvelous opportunity, and I'm grateful for the experiences I had in Norway to work with him. Dr. Robert Strangeway, of the University of California in Los Angeles, provided a lot of assistance in the statistical analysis. I enjoyed getting to know him and working with him this past year.

The KIMS instrument design is based on a design created by Jack Moore. I had the privilege of working with Jack directly when he volunteered to come and assist with the magnetic circuit assembly. His input and support was very encouraging to me at that time, and I'm grateful he was able to come and help with that part of the project. Jack passed away shortly after our encounter; he is a great scientist and a good man. I'm honored that I was able to get to know him.

My parents and family have always been there for me–ready to rejoice in successes and comfort over failures. I would not be where I am without my family.

There is one individual to whom I owe all my inspiration and motivation. In reality, I owe Him all I am. This is, after all, His work.

This work was funded in part by the NASA Graduate Student Research Program, under NASA Grant NNX08AT34H.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

`acc`	CHAMP Accelerometer Data
CDF	Common Data Format
CEM	Channel Electron Multiplier
CHAMP	Challenging Minisatellite Payload
CTS	Conventional Terrestrial System (equivalen to GEO)
`dcefs`	FAST DC Electric Field Survey
`dcmag`	FAST DC Magnetic Field Survey
DSP	Digital Signal Processor
`dsp_v58`	FAST Low Frequency, V5-V8 AC Electric Field Data
EDM	Electrical Discharge Machining
`ees`	FAST Electron Energy Spectrometer
ELF	Extremely Low Frequency
EUV	Extreme Ultraviolet
FAC	Field aligned current
FAST	Fast Auroral Snapshot Explorer
FPGA	Field Programmable Gate Array
GEI	Geocentric Equatorial Inertial Coordinates
GFZ	GeoForschungsZentrum
GHA	Greenwich Hour Angle
GSE	Geocentric Solar Ecliptic Coordinates
GSE	Ground Support Equipment
GSFC	Goddard Space Flight Center
GSM	Geocentric Solar Magnetospheric Coordinates
GST	Greenwich Sidereal Time
IC	Integrated Circuit
IDL	Interactive Data Language
`ies`	FAST Ion Energy Spectrometer
IGRF	International Geomagnetic Reference Field
ILT	Invariant Latitude
IPST	Institute for Physical Science and Technology
ISDC	Information System and Data Center (Potsdam, Germany)
KIMS	KeV Ion Magnetic Spectrograph
MCP	Microchannel Plate
MLT	Magnetic Local Time
MSP430	Texas Instruments Mixed Signal Processor (microcontroller)
NASA	National Aeronautics and Space Administration
PCM	Pulse Code Modulation
R_E	Earth radii (unit)
RENU	Rocket Experiment for Neutral Upwelling
`rso`	CHAMP Rapid Science Orbit Data
SM	Solar Magnetic Coordinates
STAR	Spatial Triaxial Accelerometer for Research
UART	Universal Asynchronous Receiver/Transmitter
UMD	University of Maryland, College Park
USB	Universal Serial Bus

Chapter 1

Introduction

This dissertation describes a study involving two phenomena observed in the Earth's atmosphere. The two phenomena, neutral upwelling and ion outflow, are similar in nature. Both are the result of heating processes in the atmosphere, and both end up in particles rising to higher altitudes within a very localized region. The goal of this study is to understand the neutral upwelling phenomenon, particularly in the context of processes similar to those that lead to ion outflow. Because both phenomena have similar characteristics and happen in the same location, the study is a search for connections that explain how these phenomena occur. Ultimately, the two processes stem from interactions at the boundary of Earth's magnetic field.

1.1 Background

The boundary between the electromagnetic fields of the Earth and the Solar System is a curious interface between the atmosphere and interplanetary space. From one aspect, the edge of the Earth's magnetic field (the magnetopause) marks a physical boundary where the solar wind is deflected around the planet. From another aspect, the boundary links the electromagnetic fields of the Sun and its solar wind with those of the Earth in such a way that motion of the solar wind leads to energy, and in some cases material, being transferred in and out of the Earth's ionosphere. These interactions lead to complex phenomena that are challenging to study, but important to understand because of their direct influence on the Earth.

Among these phenomena, heating in the ionosphere can lead to easily observable results, such as the spectacular displays of light in auroral storms, or to less obvious consequences, such as local fluctuations in atmospheric density. In some cases, heating leads to an outflow of atmospheric material from the Earth itself. One of these less visible, but influential, results is an upward movement of neutral gas within the polar cusp of the magnetosphere. Material from the mesosphere and lower thermosphere rise in this region to orbital altitudes in the middle and upper thermosphere (see Fig. 1.1).

During these neutral upwelling events, the increased density in the thermosphere causes increased drag on satellites in polar orbits as they pass through upwelling regions, slowing the spacecraft and reducing their orbital lifetimes. A unique aspect about this phenomenon is that the heating is highly localized and confined to the region around the Earth's magnetic poles known as the cusps (see Fig. 1.4). While there have been a number of clear observations of the neutral upwelling phenomenon, the mechanisms behind neutral upwelling are not yet well understood.

Considering the sharply defined location of the neutral upwelling events within the cusps of the Earth's magnetic field, we can reasonably suppose that the mechanisms responsible for lifting the neutral gas are in some way connected to the electromagnetic fields and plasmas that make up the Earth's ionosphere and magnetosphere. A number of electromagnetic and plasma effects can transfer energy to the neutral atmosphere in the cusp region, either directly or indirectly, and thus contribute to the neutral upwelling phenomenon.

1.2 Magnetospheric Physics

There are many complicated interactions in the magnetosphere, but they can be understood in terms of basic physics. Dr. E. N. Parker once described the goal of magnetospheric physics as coming to "...understand the active magnetosphere in terms of the principles of Newton and Maxwell" [1]. In other words, our goal is to describe the Earth's atmosphere and electromagnetic fields using Newtonian mechanics (particularly in terms of gravitation and fluid mechanics) and Maxwell's formulation of electromagnetism (in terms of fields, wave propagation, and plasma properties). This formulation of the physics in the magnetosphere lets us separate the region into an atmospheric and ionospheric picture. Each picture can be further described in terms of distinct layers, each with unique properties, much as the interior of the Earth can be. The atmosphere and ionosphere are very different from the lithosphere in many ways, however, and the boundaries between layers are constantly moving. While there are specific general properties in the various layers, the locations and sizes of the layers constantly vary. The dynamic properties of these different layers have consequences for radio communication, satellite trajectories, and the influence of the sun on the Earth through the solar wind.

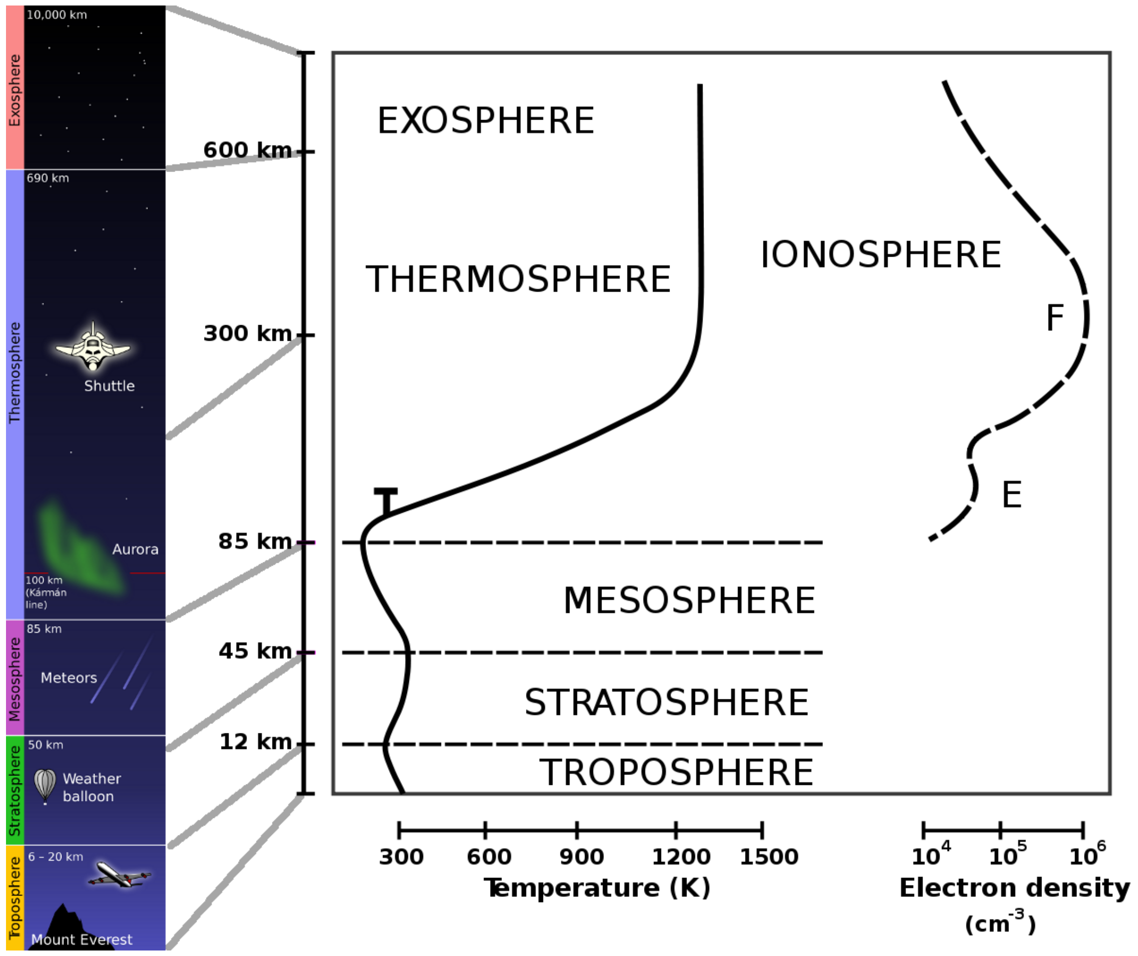


Figure 1.1: Diagram of the Earth's atmosphere. The Ionosphere varies in location and size, but typically starts in the upper Mesosphere and extends upward through the Exosphere. On the right, both pictures are depicted in terms of one of the parameters that distinguishes the layers in each. In this case, the atmospheric layers have different temperature profiles, and the ionospheric layers have different electron density profiles. (Figure credit NASA.)

1.2.1 The Atmosphere and Ionosphere

We first draw a distinction between the identified layers of the atmosphere and the ionosphere. The atmosphere refers to the different layers with distinct temperatures and compositions. We live in the troposphere, which only extends to about 0.2% of the entire height of the atmosphere, but it also contains approximately 80% of the material in the atmosphere [2]. The other layers extend upward, to an altitude of approximately 10,000 km at the exopause (see Fig. 1.1). The atmosphere, the layers that compose it, and the differing properties in each can be generally described by Newtonian physics.

The ionosphere refers to the region of the atmosphere where a substantial portion of the material is in an ionized state. Atmospheric atoms and molecules are ionized primarily by ultraviolet (UV) radiation from the Sun [3]. Like the Newtonian atmosphere, the ionosphere also has layers with distinct characteristics, such as electron density and plasma composition. The ionosphere begins in the upper mesosphere (typically about 60 km above the Earth's surface) and extends to the edge of the magnetosphere, but different conditions in the atmosphere, the sun, and interplanetary space have significant impact on locations, sizes, and even number of the ionospheric layers. Just as the atmospheric system can be described with Newtonian physics, the ionosphere can be described using a combination of Newtonian and Maxwellian (or electrodynamic) physics.

The highest well defined layer of the ionosphere is referred to as the F layer. The F layer is of particular interest because peak plasma density occurs there.

This layer is generally between 200 and 500 km above the Earth's surface, in the atmospheric layer called the thermosphere. It is in this region that neutral upwelling occurs. Beyond the F layer lies the topside layer, with gradually falling density and increasing ionization fraction. The topside layer extends through the exosphere and out to the magnetopause.

A number of important parameters describe the atmosphere and ionosphere. One of the more important of these is the scale height. This parameter is a measure of how quickly a value decreases with height, and in terms of the atmosphere is derived as a description of the atmospheric pressure. The atmospheric scale height is a ratio of thermal energy to gravitational potential energy. For a given temperature, the scale height is given by

$$H = \frac{kT}{Mg} \tag{1.1}$$

using Boltzmann's constant k, the temperature T, the molecular mass of air M, and gravitational acceleration g. For a given height above the surface z and base surface pressure P_0, the pressure is then approximately

$$P = P_0 \mathrm{e}^{-\frac{z}{H}} \tag{1.2}$$

As the temperature increases, so does the scale height, and thus the pressure at a given altitude. We then expect the density of the atmosphere at higher altitudes to increase because of an increase in temperature. Processes that heat the atmosphere will increase scale height. As such, scale height varies with time of day, day of year, latitude, and a number of other parameters that influence the local temperature of the atmosphere. The Extended Mass Spectrometer–Incoherent Scatter (MSISE)

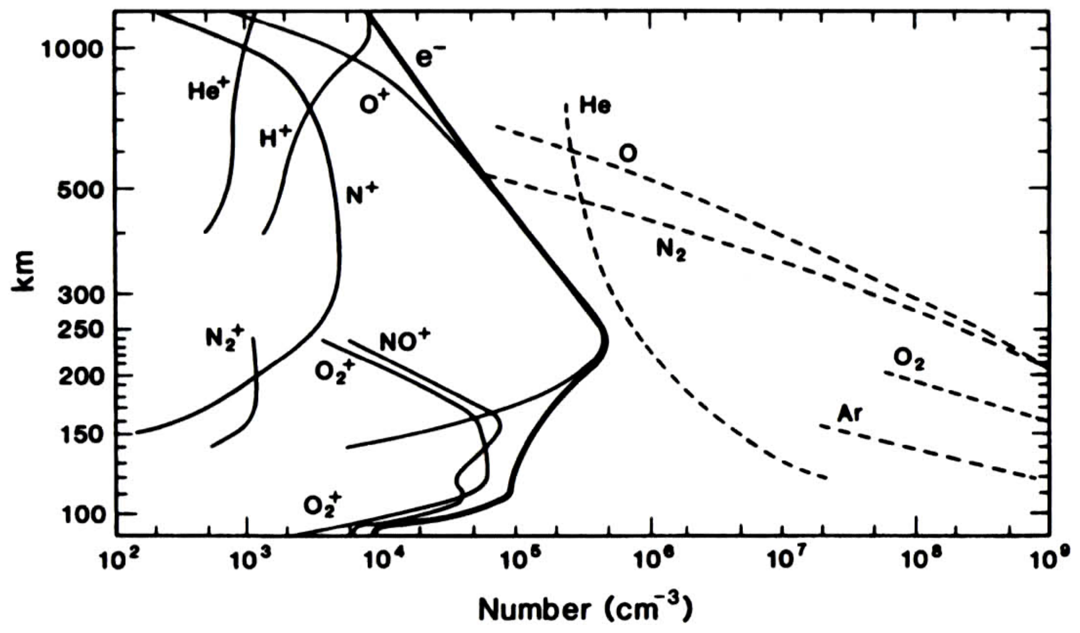


Figure 1.2: Plot showing the densities of various components of both the ionosphere (solid lines) and the atmosphere (dashed lines). The neutral density in the F layer is typically 1–3 orders of magnitude greater than the plasma density. These data are based on multiple mass-spectrometer measurements compiled for the international quiet solar year in 1969 [6].

atmospheric model [4], which is based on measurements of many different atmospheric and solar parameters, estimates a typical scale height for the Earth's lower atmosphere of about 6.7 km [5].

Density is a very important parameter for both the atmosphere and the ionosphere. Even at the same altitude, however, both environments can have remarkably different densities and compositions (see Fig. 1.2). At orbital altitudes multiple scale heights above the Earth, the density of the neutral atmosphere can be orders of magnitude greater than the density of the ionospheric plasma.

Related to density is the mean free path. The mean free path is a description of length scales between particle interactions, typically describing how far a particle may travel before colliding with another particle. This value can be described in terms of the density n and the cross-section area for collisions σ by

$$\lambda_{mfp} = \frac{1}{\sigma n} \tag{1.3}$$

The cross-section σ depends on a number of parameters, including the temperature and composition of the ionosphere in the region of interest. A rough estimate can be found based on the temperature and density of the electrons, as given by Alfvén [7]:

$$\lambda \sim 10^4 \frac{T_e^2}{N_e} \tag{1.4}$$

For ionospheric plasma in the F region, we find temperatures between 1000 and 2000 K, and densities peaking at 10^6cm^{-3}. These values suggest a mean free path in the ionosphere of tens to a few hundreds of kilometers. This length is comparable to the mean free path of a confined gas with a density approximately that of the neutral atmosphere at these altitudes, suggesting an overall pressure below 10^{-6} torr in this region.

1.2.2 The Magnetosphere

The Earth's magnetic field creates a third system within the overall region of interest. The magnetic field interacts with the interplanetary magnetic field (generated by the Sun) and with the solar wind, creating a shell that surrounds the Earth to an extent of about 15–20 Earth radii (R_E). The solar wind compresses the front of the shell to about 10–12 R_E while the back is extended into a long "magnetotail", reaching beyond 200 R_E (see Fig. 1.4). We call the boundary of this

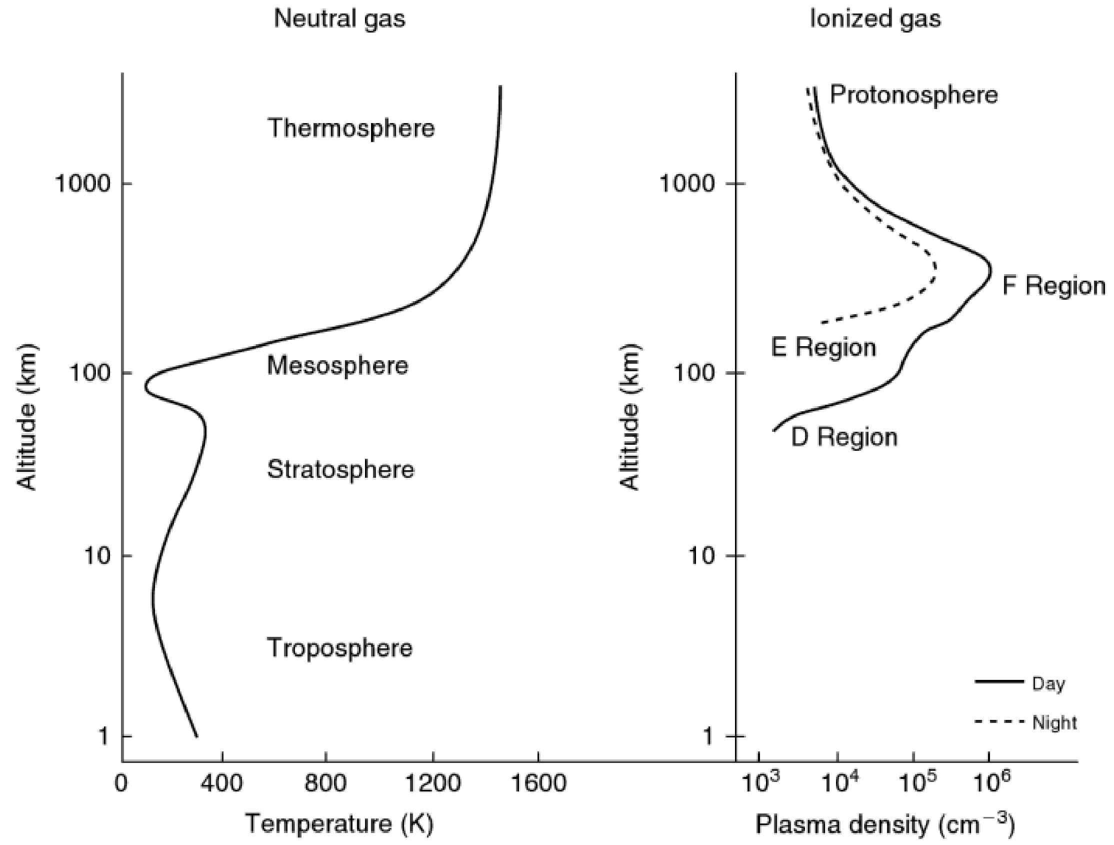


Figure 1.3: Diagram showing a configuration of Earth's ionosphere in relation to the atmosphere. Conditions present during the daytime can also lead to another layer (the D layer) at lower altitudes, and at times a division is seen in the F layer (the two parts being called the F1 and F2 layers). (Figure credit M.C. Kelley [3].)

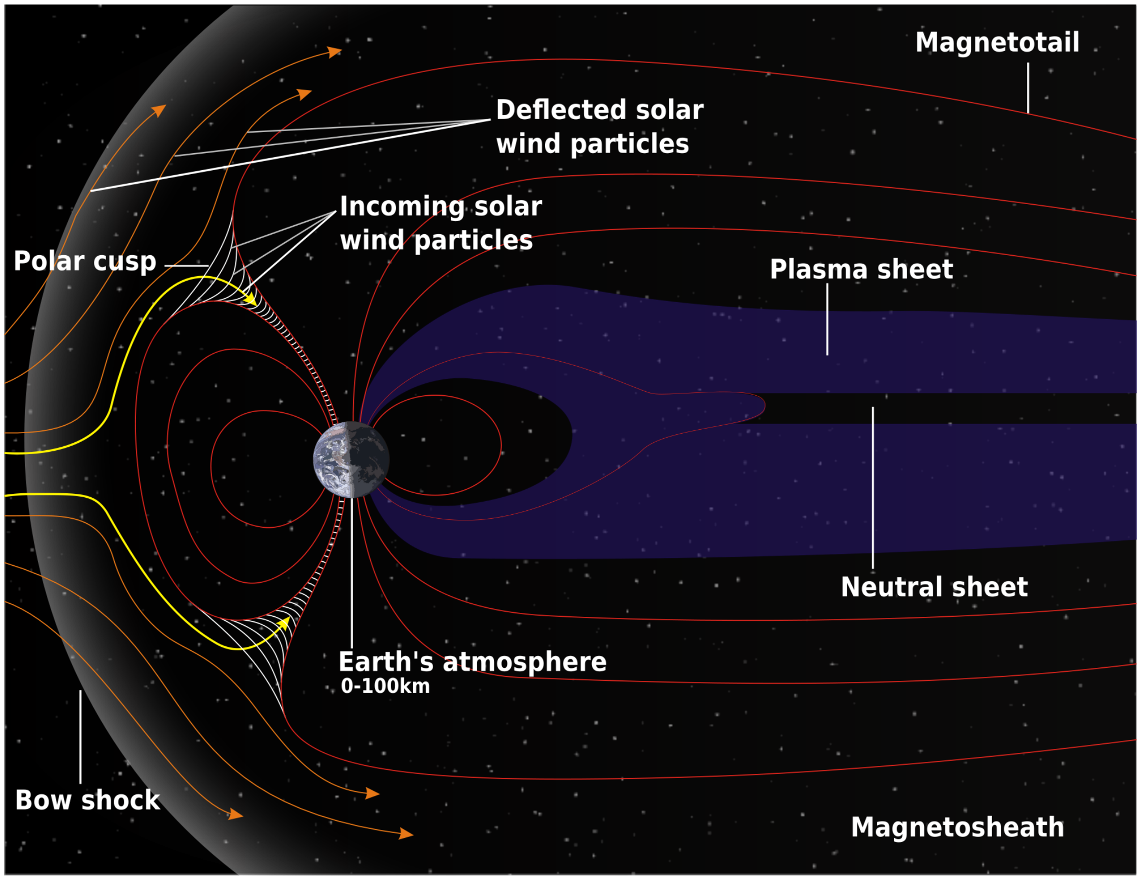

Magnetotail
Deflected solar wind particles
Incoming solar wind particles
Polar cusp
Plasma sheet
Neutral sheet
Earth's atmosphere 0-100km
Bow shock
Magnetosheath

Figure 1.4: Diagram of the Earth's magnetosphere with key features identified. The bow shock is where the solar wind slows to below the local sound speed, creating a shock wave as the solar wind is deflected around the magnetosphere. The movement of the solar wind around the Earth draws the magnetic field lines out to create a long Magnetotail. The funnel shapes formed at the magnetic poles are called the polar cusps. Solar wind particles can pass into the cusps and precipitate into the atmosphere. Atmospheric particles can escape through the cusps. The plasma sheet, neutral sheet, and magnetosheath are all aspects of Earth's magnetosphere that result from interactions of charged and neutral particles with the magnetosphere, but do not enter this study. (Figure credit NASA.)

shell the magnetopause.

Earth's field is roughly dipolar, with an average strength of approximately 50 μT at the surface. It is currently believed that the field is created by electric currents in the outer core according to the dynamo theory[1]. The north and south poles of Earth's magnetic field shift in location independently, and both poles have drifted substantially (on the order of $\sim$100 km over the past 100 years [55]. Nevertheless, the magnetosphere can be modeled to fair accuracy as a magnetic dipole with a few small corrections in higher order terms of a multipole expansion. Standard models of the Earth's magnetic field include the International Geomagnetic Reference Field (IGRF) [9]. The IGRF is often used to evaluate atmospheric and electromagnetic phenomena as they relate to local changes in the field.

Of particular interest to the neutral upwelling phenomenon being discussed are the funnel-shaped regions of the magnetic field found at the magnetic poles. These regions, called the cusps, allow for energy (sourced from both fields and particles) to be exchanged between the interplanetary medium and the ionosphere [10]. The nature of this exchange is an ongoing topic of interest in space physics. The cusps are surrounded by the annular shaped auroral regions, where solar wind interactions become visible through spectacular displays of light in the atmosphere commonly known as the northern lights.

[1]Dynamo theory suggests that the rotation and convection of an electrically conductive fluid, such as the liquid outer core of the Earth, can sustain magnetic fields for astronomical time scales, and thus may be the source of planetary and stellar magnetic fields. The theory was first suggested by Larmor [8].

1.2.3 The Solar Wind

The Sun, due to the heat and pressure of the plasma in the corona, drives material away from itself into interplanetary space. This "solar wind" is made mostly of electrons and protons, and is accelerated away from the sun by both thermal and magnetic forces (according to current theories) [11]. Particles able to escape the Sun's gravitational pull must have substantial kinetic energy, and in fact the solar wind is made of energetic particles with velocities of typically 400 to 650 km/s. (For protons, this typical velocity is equivalent to energies on the order of a few keV per nucleon.)

The solar wind expands through the solar system, interacting with the planets, moons, comets, and other various objects of the heliosphere (including man-made objects). The effect the solar wind has in these encounters depends on the nature of the object; it is responsible for the tails in comets, for example. Other bodies with atmospheres directly impacted by the solar wind would have that atmosphere stripped away in like manner [12]. The solar wind would have a similar effect on our own planet, were it not for the Earth's magnetic field.

The Earth's magnetosphere acts as a shield from the solar wind. When electrons and ions in the solar wind encounter the Earth's fields, the Lorentz force[2] causes the particles to slow and deflect around the outside of the field. The resulting shock wave[3] that results where the solar wind directly impacts the magnetic

[2]The Lorentz force describes the forces felt by charged particles moving in a magnetic field, where $F = q\vec{v} \times \vec{B}$.

[3]A shock wave occurs when the velocity of the particles rises above or drops below the local

field and slows down is called the bow shock (see Fig. 1.4). Through this interaction, the solar wind particles are heated to energies ranging from 1–10 keV. Most of the solar wind is deflected completely around the Earth, which has the effect of creating the long tail characteristic of the Earth's magnetic field. The compression that occurs at the bow shock is dependent on the exact conditions of the solar wind. During periods of substantial solar activity, the Earth's magnetic field can compress far enough to expose orbiting satellites directly to the solar wind. Though for the most part the solar wind material is deflected around the Earth, some of the material precipitates into the Earth's atmosphere through the cusps, resulting in direct interaction between solar and ionospheric material.

1.2.4 Ionospheric Plasma

A description of the ionospheric plasma depends on the specific region of the ionosphere. In the topside layer, where much of the direct interaction with the solar wind takes place, most of the ionospheric plasma is ionized oxygen atoms. The protons of the solar wind penetrate through the topside layer, and at lower altitudes account for a substantial portion of the plasma composition. At these altitudes, we also find ionized molecular oxygen (O_2^+), and at the lowest altitudes of the ionosphere we also find ionized nitric oxide (NO^+). Each population of the ionospheric plasma has different properties, such as energy and density. Though there are potentially many constituents to the ionospheric plasma, within the region of the polar cusp at

sound speed. The sound speed in a plasma is given by $c_s^2 = \gamma p/\rho$, where γ is the ratio of specific heats, p is the plasma pressure, and ρ is the plasma density.

orbital altitudes the dominant ions are oxygen ions (O^+) at energies of up to 10 eV and 1–10 keV solar wind protons.

The ionospheric plasma is quasi-neutral[4], so a complete description should also include the electron population. In the F layer, the electron density (and by quasi-neutrality the ion density) will typically be 10^5–10^6 cm^{-3}. Electrons released from photo-ionization in the ionosphere typically have thermal energies of up to a few eV [13], while the hotter solar wind electrons have thermal energies of a few 10's of eV and can be accelerated to 100's of eV [14].

1.3 Heating in the Ionosphere

The transfer of energy from outside the magnetosphere to the ionospheric plasma can result in an increase in temperature of either ions or electrons within the plasma, the neutral gas of the atmosphere interspersed with the plasma, or any combination of the particle states. In fact, heating of one component can result in heating of the others through common interactions. In this way, the neutral gas can be heated both directly and indirectly by heating processes that occur in the ionosphere.

There are a number of heating processes that can occur in the ionosphere. While most of these are similar in the means of heating the neutral gas in the atmosphere, the subtle distinctions between them give information about the initial

[4]Quasi-neutrality is an assumption often made in plasma physics that the number of positive charges and the number of negative charges within an arbitrary volume of plasma are roughly equal.

causes and sources of energy for the heating. In all cases, heating of the neutral gas causes it to expand, transporting neutral gas to higher altitudes. This rising, or up-welling, of the neutral gas increases the density at higher altitudes while decreasing the density at lower altitudes.

The heating processes that are considered most important are electrodynamic in nature; while extreme ultraviolet (EUV) light from the sun does heat our atmosphere, magnetospheric inputs dominate the energy transfer to the neutral gas in the ionosphere [15]. These inputs are the result of an interaction between ionospheric plasma (created by solar EUV ionizations in the ionosphere) and the solar wind.

1.3.1 Joule Heating

When the solar wind passes over the Earth's magnetosphere, it produces a current at the magnetopause. The presence of the current links the solar and terrestrial magnetic fields, allowing energy to be transferred from the solar wind to the ionosphere by induction. The electric fields created between the solar wind and the ionosphere drive currents in the ionospheric plasma perpendicular to the magnetic field. Since the neutral gas density is much larger than ion density in the ionosphere, the ionospheric plasma dragged along by the solar wind currents pass through a collisional neutral gas. The friction created by collisions between the ions and the neutrals transfers energy from the solar wind to the neutral gas and causes it to heat. We refer to this specific heating process as Joule heating.

Joule heating is often seen when an increase in activity between the solar wind and the ionosphere causes a temporary perturbation in Earth's magnetic field over a large area. These events are called geomagnetic storms. In typical storms, the peak Joule heating rate occurs near 150 km in altitude [16].

1.3.2 Ohmic Heating

As the solar wind passes over the Earth, the subsequent movement of the ionospheric plasma can distort the Earth's magnetic field locally. Distortion of the magnetic field causes tension-like forces on the plasma. If the tension created by the distortion and the forces due to the pressure of the plasma itself are unbalanced, currents will flow along magnetic field lines. These so-called field aligned currents (FACs) of ionospheric electrons and ions increase the collision rate between the charged particles, heating the ionospheric plasma in much the same way a resistor in an electric circuit is heated when current passes through it. When the electrons and ions upwell due to the increase in temperature, neutral gas is dragged along with the rising plasma, causing neutral upwelling.

Though the overall effect is the same as in Joule heating, where the heat transferred to the ionosphere causes the neutral gas to be heated and upwell, the distinction is made here that the collisions are primarily between electrons and ions rather than ions and neutrals. To distinguish this process from Joule heating, this process is called Ohmic heating to be consistent with literature discussing electron heating and cooling processes [17].

1.3.3 Electron and Ion Precipitation

Under certain conditions, where the solar and terrestrial magnetic fields are aligned favorably for magnetic reconnection[5] to occur, the particles in the solar wind are able to penetrate the magnetopause at the cusp. These particles are sometimes responsible for the aurora, and can also heat the ionosphere.

Since the typical temperature of ionospheric plasma is on the order of 1–10 eV, mixing of solar wind plasma of temperature 1–10 keV with the ionosphere increases the overall temperature of the electron or ion plasma. An increased number of charged particles can also lead to increased Ohmic heating. The resulting electric fields from the movement of the heated plasma particles cause the ionospheric plasma to expand, again dragging the neutral gas along. This mechanism is distinct in that the source of energetic particles is the solar wind itself, and not the ionosphere.

1.3.4 Alfvén and Other Plasma Waves

At high altitudes, the ionospheric plasma is turbulent. In this turbulent region, a number of types of plasma waves can occur. In addition, small-scale field aligned currents can drive waves in localized regions of ionospheric plasma, particularly in the auroral zone. The energy in these waves can locally heat the ionosphere through

[5]Magnetic reconnection describes an interaction of a plasma as it moves between magnetic field regions with different topologies, such as the interface between the interplanetary magnetic field and the Earth's magnetosphere. Reconnection causes charged particles to accelerate outward from reconnection points. Southward pointing interplanetary magnetic fields are most favorable for reconnection at the dayside magnetosphere. See, for example, [18] for further information.

Joule and Ohmic heating. A key distinction in plasma wave heating is that it is very localized rather than heating broadly over the ionosphere.

1.4 Consequences of Ionospheric Heating

1.4.1 Upwelling

Whenever a gas is heated at constant pressure, it will expand. The heating that occurs in the ionosphere has the same effect. When the heating is localized within a particular altitude, the atmosphere expands upward. The expansion decreases gas density at that altitude, and increases density at the altitudes to which it expands. The resulting density enhancement is what we call upwelling. The upwelling phenomenon can occur for both neutral gas and plasma.

1.4.2 Jeans Escape

The atoms, molecules, and ions in our atmosphere are confined by Earth's gravity. Just as a rocket with enough velocity can escape the gravitational field, so can any of these particles. If there is sufficient heating, a particle can accelerate to a velocity above the escape velocity, and the particle can leave the atmosphere. This mechanism is often called Jeans escape, named after Sir James Jeans, who first described the process. The escape velocity, given by:

$$v_e = \sqrt{\frac{2GM}{r}} \tag{1.5}$$

depends on the mass of the Earth M and the distance from the center of the Earth r. At orbital altitudes, this value is roughly 10 to 11 km/s. For hydrogen, that velocity is equivalent to an energy of only 0.5 eV, which is why we see so little hydrogen naturally occuring in our atmosphere.

For the common constituents of our neutral atmosphere, heating processes could feasibly cause some of the gas to escape Earth's gravity and flow into space. The amount of heating we observe, however, is not sufficient to cause significant loss of our neutral atmosphere.

1.4.3 Ion Outflow

While there is not a substantial loss of neutral gas through Jeans escape, there are periods of sufficient heating, particularly in the cusp, of ions such that they are able to reach escape velocity. For oxygen atoms, the equivalent energy to escape velocity is 10 eV, low enough to be achieved through the heating processes we have described. A loss of ionospheric ions does occur under certain circumstances, and a flow of ions from the polar cusps with energies $\sim$10 eV has been observed. This phenomenon is called ion outflow.

1.5 Measurements of Ionospheric Heating

1.5.1 Neutral Upwelling

The phenomenon of neutral upwelling in the cusp has been observed since early in the space age. In 1963, the Injun-3 satellite experienced an unexpected increase in drag when traveling through the auroral zones [19]. The increased drag was attributed to an increased ionosphere density in that part of its orbit. Injun-3 orbited at an altitude of only 250 km. Joule heating processes in the stratosphere and ionosphere seemed the likely source of the increase in neutral gas density.

In 2000, the CHAMP satellite was put into a circular, polar orbit at approximately 400 km altitude. CHAMP's suite of instruments includes a very precise and sensitive 3-axis accelerometer that takes measurements on a much shorter time base (typically 10 s) than an orbital period. The high resolution data it provides are useful in resolving features in the atmosphere causing drag on the satellite. Using data from this instrument, Lühr *et al.* [20] observed that density enhancements could be as much as a factor of two greater than normal, and that the enhancements were highly localized to the cusp region. These enhancements were coincident with large field-aligned currents in the ionosphere.

More recently, a study using neutral density data from the STREAK mission found a density depletion, rather than an enhancement, at an altitude of 250 km in the south magnetic cusp [21]. The depletion was only 1–2% of nominal densities, however in the context of the results of Lühr *et al.*, the results suggest that there is something causing the neutral density changes other than (or perhaps in addition to) Joule heating as previously assumed.

One assumption that explains the observed upwelling is that more Joule heating occurs than was expected. However, recent modeling suggests that much stronger Joule heating still is not sufficient to cause the observed upwelling, and does not explain its distinctive localization [22],[23]. Other possible contributors to neutral

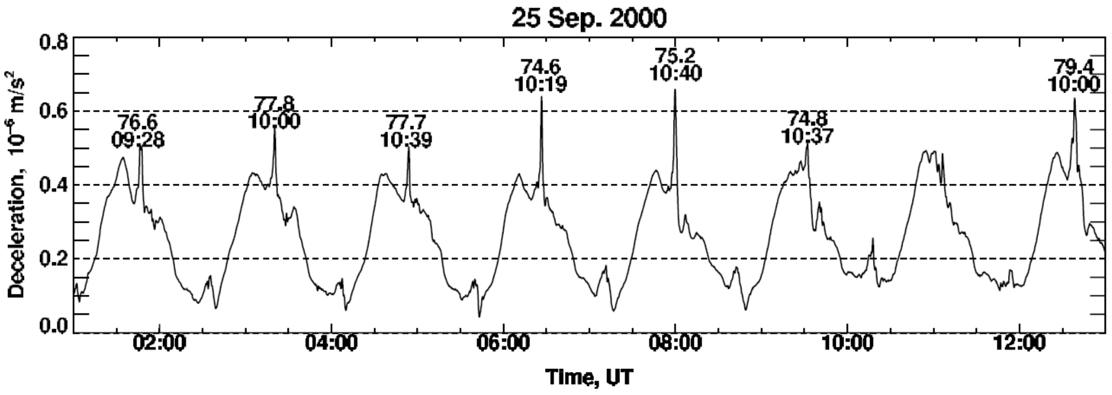


Figure 1.5: Results from Lühr (2004) [20] showing the deceleration of the CHAMP satellite over multiple orbits on 25 Sep 2000. The results show significant density enhancements at high invariant latitude near magnetic noon, corresponding to the spacecraft passing through the cusp. Invariant latitude (ILT) of the spacecraft and magnetic local time (MLT) are labeled at each peak. ILT and MLT are essentially the equivalent of latitude and longitude in a coordinate system for Earth's magnetic field. (See Sec. 4.7.5.)

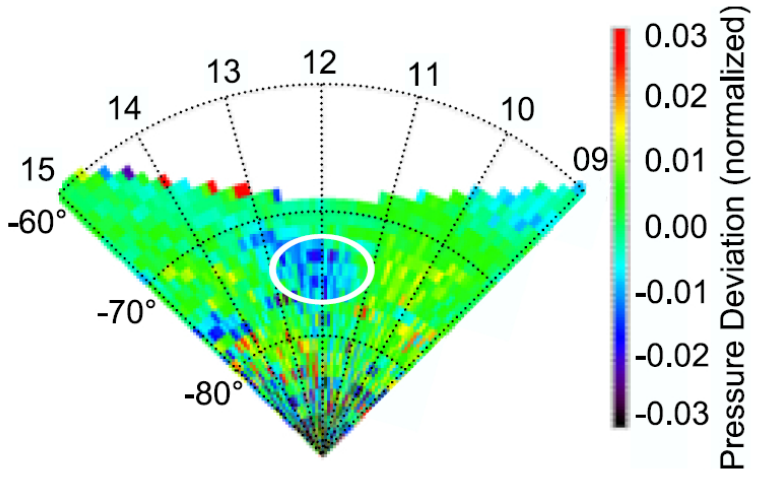


Figure 1.6: Results from Clemmons (2008) [21] showing a depletion in neutral density at altitudes where enhancement should be seen assuming Joule heating accounts for the upwelling. Coordinates are in ILT (radial) and MLT (angular), or latitude and longitude coordinates in the Solar Magnetic coordinate system (see Section 4.7.5). The white oval suggests the approximate location of the day-side cusp, and matches the location of the depletion.

upwelling include Alfvén waves in the aurora (a potential source of extra Joule heating), small and large scale field aligned currents, and electron precipitation.

1.6 This Study

1.6.1 RENU

To better understand the neutral upwelling phenomenon and the processes that ultimately drive the heating of the neutral gas, a sounding rocket campaign was proposed and carried out. The Rocket Experiment for Neutral Upwelling (RENU) is a collaboration of six research groups that designed a sounding rocket payload consisting of 14 separate instruments. The payload was designed to launch on a Black Brant XII rocket and be carried to an altitude of 500 km over the north magnetic pole.

Ultimately, the RENU campaign was unable to obtain the needed data due to a malfunction of the nose cone ejection system. The project is currently under consideration for refunding to launch again.

Our group at the University of Maryland, College Park (UMD) and NASA's Goddard Space Flight Center (NASA/GSFC) designed, built, and tested one of the instruments on board the RENU payload. The keV Ion Magnetic Spectrograph (KIMS) instrument measures high energy ion distributions between energies of 0.1 and 10 keV. Chapters 2 and 3 of this dissertation describe the KIMS instrument and its testing. The design of the instrument developed new techniques for measuring ion energy distributions that are effective for sounding rocket flights, not the least of which is a new, inexpensive fast amplifier system for microchannel plate detectors. The mechanisms and design parameters of the KIMS instrument will be described in Chapter 2, and the testing results in Chapter 3.

1.6.2 Statistical Analysis of Ionospheric Energy Sources

With the loss of the RENU payload, another study was undertaken to better understand the role electrodynamic energy sources play in neutral upwelling. This analysis used data from the Fast Auroral Snapshot Explorer (FAST) in addition to CHAMP data from the Spatial Triaxial Accelerometer for Research (or STAR accelerometer) to examine the statistical correlations between electromagnetic (Poynting) energy flux and kinetic (particle) energy flux. As a statistical study, no direct causal relations for the causes of neutral upwelling can be drawn from the correlations, but we can examine the relative importance of these energy sources. This analysis identifies the properties most useful to study when a second RENU-like payload is prepared. Further background for this study is provided in Chapter 4, a description of the analysis methods in Chapter 5, and the results of the analysis in Chapter 6.

Chapter 2

Ionospheric Ion Detection and the KIMS Instrument

The environment in the ionosphere and in space is hostile and unforgiving to instruments of any sort. Any instrument designed to make measurements in these regions is a collection of compromises. The goal in developing an instrument to measure fields or particle properties in space is to sense the properties needed without affecting the environment itself. In reality, any detection scheme will exert some influence on its surroundings at the detriment of accuracy in measuring the properties of interest. For particles, such as ions, instruments are designed to collect ions from the surroundings in such a way as to be able to determine their sources and their energies. In some cases, the instrument can also identify the different ion species present.

Over the years, a number of different detection schemes have been developed. Each has its advantages for certain types of measurements and certain circumstances. Many of these designs use electric fields to select specific energies per charge or, in some cases, direct particles to a specific detector. These designs often require sweeping voltages in order to scan energy per charge and/or viewing angles. Such a detection scheme is termed a spectrometer, as it can only measure one value of the spectrum (be it energy or direction) at a time. There are also detectors that are able to measure a range of values simultaneously. Devices that are able to capture a range of values of a spectrum are termed spectrographs.

Descriptions of many successful ion and electron instrument designs that have flown on important and historical missions have been collected and compiled for reference [24]. The articles in this compilation help to understand how the various types of spectrometers and spectrographs were built and how they function. Though these instruments are very useful and well-designed, new instrument designs make it possible to examine new environments and phenomena in unique ways.

An ion spectrograph that takes measurements of the local charged particle energy spectrum at millisecond time resolution is ideal for a study of the neutral upwelling phenomenon. On a sounding rocket flight, mass is often not so much of a consideration as on orbiting spacecraft, and so a permanent magnetic field can be used to separate ions of different momenta in the instrument rather than using a sweeped voltage to scan ion energies. Aside from the mass of permanent magnets and its supporting structure (the yoke), this scheme has the disadvantage of affecting the magnetic fields in the region around the instrument. Materials with high magnetic permeability and high magnetic saturation points are used to shield other instruments and the local plasma external to the instrument from the strong internal fields. In this chapter, we describe a spectrograph designed for the RENU sounding rocket campaign. This instrument is able to measure ions with energies between 0.1 and 10 keV (assuming singly charged hydrogen ions) at a rate of 250 Hz. The instrument design is called the KeV Ion Magnetic Spectrograph (KIMS).

2.1 The KeV Ion Magnetic Spectrograph

The KIMS design makes use of two permanent magnets in a high magnetic permeability yoke. The yoke is specially designed to provide two regions of constant magnetic field. The magnets and yoke assembly are referred to as the magnetic circuit. Inside the constant field regions, ions undergo cyclotron motion[1]. The radius of the circular ion trajectories is proportional to the momentum per charge of the ion. In this way, the magnetic field effects a spatial separation of ions according to their momentum per charge. In the Earth's cusp, the bulk of the ion composition is singly-ionized hydrogen atoms (protons) from the precipitating solar wind, and so, to a reasonable approximation in this region, the momentum distribution is equivalent to an energy distribution of the ionospheric protons.

2.1.1 Energy Spectrum Dispersion

The predicted energy distribution for the design can be modeled simply. Since cyclotron motion is circular, we can find the radius of rotation simply:

$$r_c = \frac{v}{\omega_c} = \frac{mv}{qB} = \frac{\sqrt{2mE}}{qB} \tag{2.1}$$

By placing a detector in the same plane as the instrument entrance aperture, particles will be detected after half of a cyclotron rotation. The position on the detector where an ion will land is one diameter from the entrance point, or, accounting for a

[1] Cyclotron motion refers to the gyration of charged particles in a magnetic field. The Lorentz force $q\vec{v}\times\vec{B}$ causes charged particles moving perpendicular to the magnetic field to follow a circular path (helical if the velocity has a component parallel to $\vec{B}$).

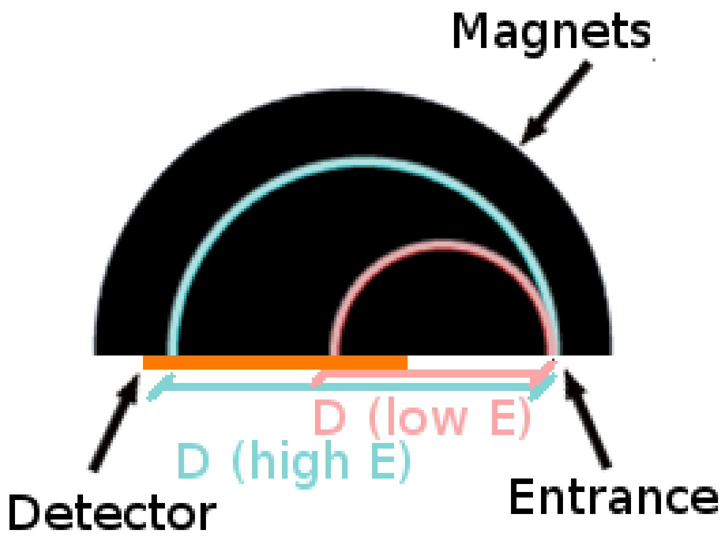


Figure 2.1: Diagram showing the motion of ions in the KIMS magnetic field. Ions with more momentum (or energy for like-species) per charge have a larger radius, intersecting the detector farther from the entrance aperture.

finite width aperture,

$$d = 2r_c + s = \frac{2\sqrt{2mE}}{qB} + s \tag{2.2}$$

where s is the entrance point measured from the center of an aperture of width w with s positive when located closer to the detector than the aperture center. With these conventions, the value d is the distance from the aperture center to the impact point, providing a useful reference from which to measure the detector position.

2.1.2 Field of View

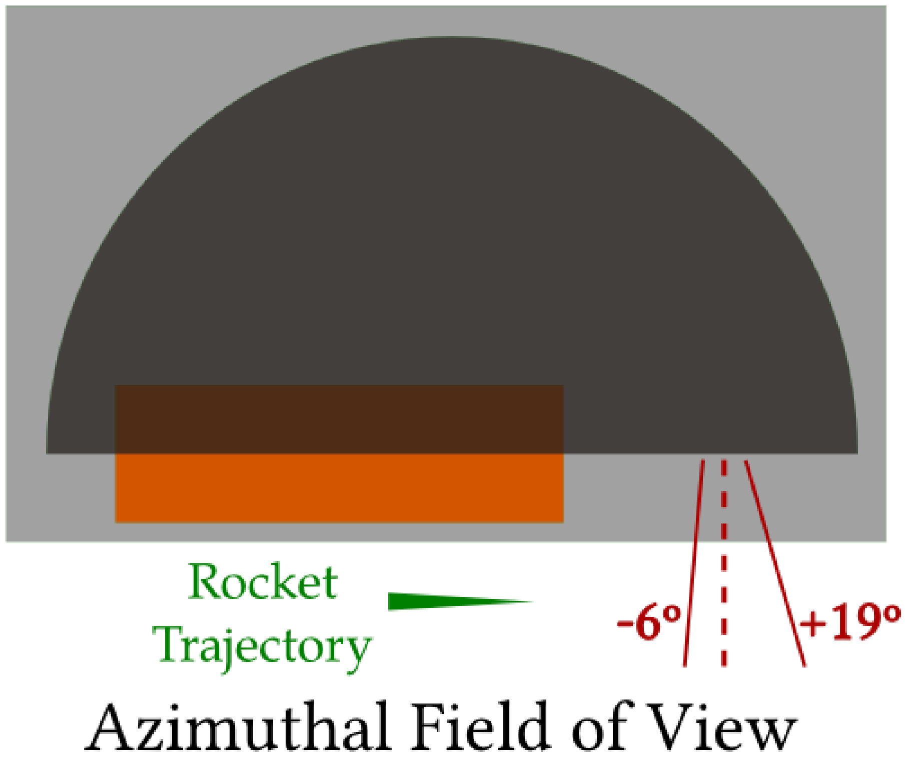
In reality, ions do not always enter the instrument at normal incidence. The incidence directions to this instrument will be described as azimuthal (angles away from normal parallel to the planes of the magnet surfaces, but still perpendicular to the magnetic field) and perpendicular (as in perpendicular to the magnet surface plane, but parallel to the magnetic field, see Fig. 2.2). A more general expression for the trajectory that accounts for the azimuthal angle of incidence is

$$d = 2\cos\theta\frac{\sqrt{2mE}}{qB} + s \tag{2.3}$$

where θ runs between the angular limits on the instrument's view with $\theta = 0$ implying normal incidence. The KIMS instrument is designed with an azimuthal field of view that ranges between +19° and -6°. KIMS is mounted on the RENU payload with the azimuthal directions pointing in an arc along the direction of movement of the rocket. This results in a field of view looking radially outward and slighty elevated to the payload trajectory. Since the trajectory of RENU is designed to be aligned with the local magnetic field, ions travelling in directions nearly perpendicular to the local magnetic field are measured by the KIMS instrument.

The instrument also has a finite acceptance angle perpendicular to the plane of the magnets. Cyclotron motion only results from the perpendicular component of the ion's velocity, so the ions are distributed according to their perpendicular momentum. The effect of nonzero angles in this perpendicular direction is to reduce the observed energy of the particle. Since nonzero perpendicular motion implies motion parallel to the magnetic field, the velocity component in this direction is not subject to magnetic forces, and the cyclotron motion will be determined by the remaining component perpendicular to $\vec{B}$.

The height h of the spaces between the magnets and the partition of the



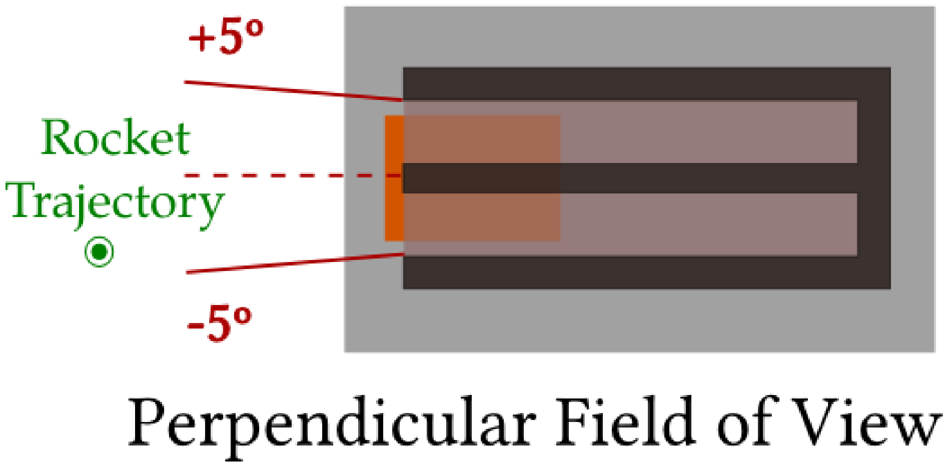


Figure 2.2: Field of view for the KIMS instrument. Azimuthal direction looks upward, between +19° and -6°. Perpendicular direction looks radial, limited by the mechanical design to between ±5°.

instrument defines the perpendicular acceptance angle ϕ, as given by

$$|sin\phi| < \frac{h}{\pi}\frac{qB}{mv} \tag{2.4}$$

At large enough angles, ions will no longer be detected as they will collide with the magnetic circuit before reaching the detection plane. For increasing energy, the angle at which this occurs decreases with $\frac{1}{\sqrt{E}}$. At small vertical angles, the ion velocity component parallel to $\vec{B}$ is negligible. Small, finite acceptance perpendicular to the plane of the magnets, then, only limits the view of the instrument and does not have an appreciable effect on the energy response. In KIMS, the height of each partition is approximately 12.7 mm. With the given magnetic fields of 0.1 and 0.27 T and measured energies of 0.1 to 10 keV, the perpendicular angular acceptance must be less than 16° with a corresponding reduction in measured energy of less than 8% (see Fig. 2.3). At this angle, only the lowest energy ions (∼100 eV) are detected, and so the mechanical structure of the instrument aperture further limits this acceptance to 5° on either side. This limit lowers the effective measured energy reduction to at most a negligible 0.8%.

2.2 Microchannel Plates

The detector used in the KIMS instrument consists of a pair of microchannel plates (MCPs) [25]. The MCP is a matrix of narrow glass tubes, or channels, sliced into thin wafers. Each face of the wafer is coated with a conductive material. An incident particle of sufficient energy will cause electron emission from the semiconductive wall of the channel struck by the particle. When a strong electric field is

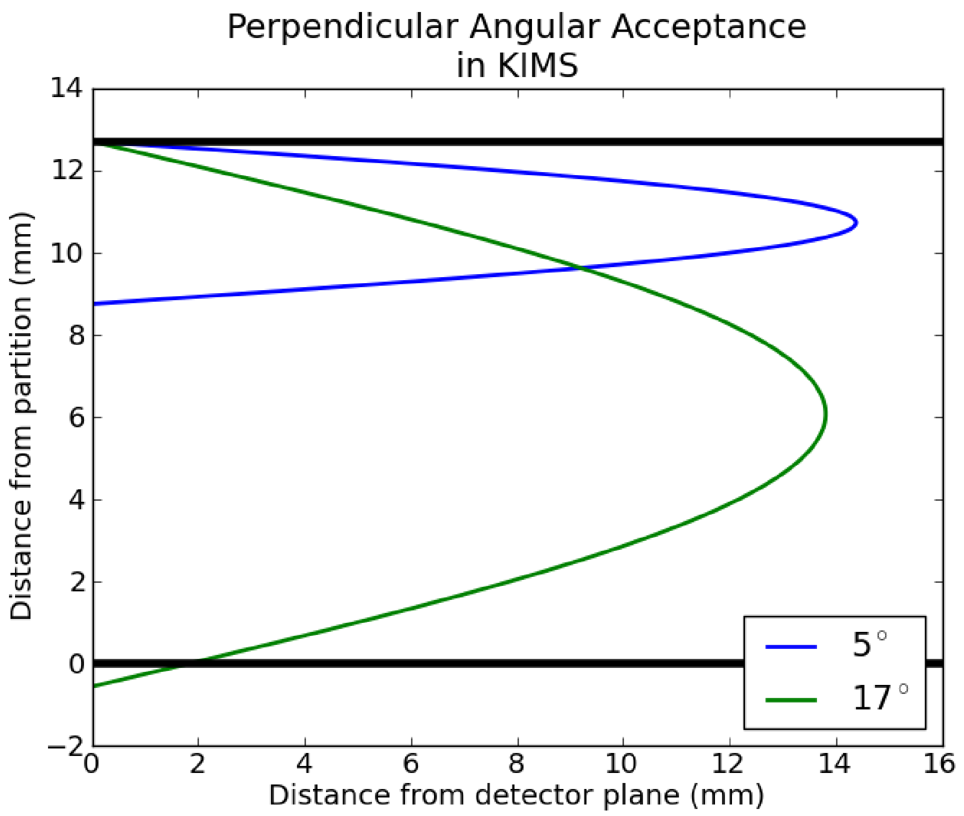


Figure 2.3: Particles can only reach the detector in KIMS if they enter the instrument within a finite field of view in the perpendicular direction. A particle entering at 5° will reach the detector with a negligible difference in its effective energy. A 100 eV particle entering at 17° would have a more noticible energy reduction, but would collide with the magnetic circuit's partition before reaching the detector. Higher energy ions collide at lower angles. In KIMS, the mechanical structure holding the magnetic circuit limits the view to ±5° in the perpendicular direction.

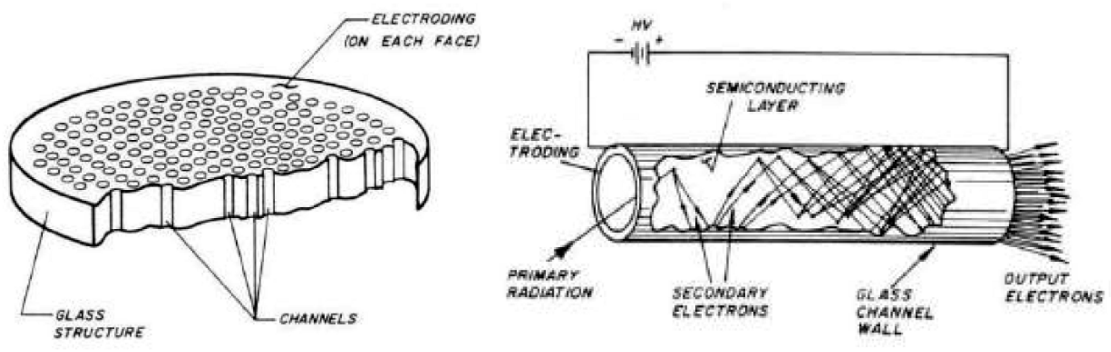


Figure 2.4: Diagram of MCP function, taken from Wiza [1979].

created by applying an electric potential across the plate, the secondary electrons are accelerated until they collide with the channel wall themselves, producing additional secondary electron emissions. The end effect is an avalanche of secondary emissions as each newly emitted electron is driven further down the channel by the electric field. The result of a single particle incident on the MCP is an emission of a large number of electrons, usually on the order of 10^3. A second plate added to the system increases the total gain to 10^6, providing easily detectable current pulses for single particle events.

Output current pulses from the MCP are collected with a segmented anode. A circuit board is etched to leave a number of exposed pads located behind the MCP output. The charge pulses are collected by the plates and amplified by a charge sensitive amplifier that triggers a logic pulse from a discriminator. The logic pulses are counted to record the number of events that occur at a given pad over a given time. Since electron emission in the MCP is a statistical process, a varying number of electrons can be emitted for each event. The resulting distribution of charge pulse amplitudes are commonly known as the pulse height distribution (PHD). The pulse height distribution of an MCP can be described as a negative exponential when the plates are not saturated [26], and have been found to be a property of the plates themselves, being nearly independent of the type of incident particles [27]. These properties simplify the detector, as its calibration requires only an appropriate threshold for counted pulses rather than a full pulse height analysis [28]. This threshold is selected to separate the PHD from the background noise in the detector electronics.

2.2.1 KIMS Detector Response

For a given momentum and angle of incidence, a uniform distribution of ions across the aperture width will map to a region of the same width at the detector plane. This creates a nearly triangular probability distribution for the energies of particles detected by any given pad on the segmented anode for a given angle of incidence (see Fig. 3.3). Including all angles of incidence creates a more Gaussian distribution. A Monte Carlo simulation including a range of energies, aperture positions, and angles of incidence provided a model of the energy response function for the instrument design. (See Section 3.1.1 for more details.)

2.3 Project Scope

The concepts behind the design of KIMS are general and can be adjusted to suit the needs of a variety of enivronments. The design of the KIMS instrument was

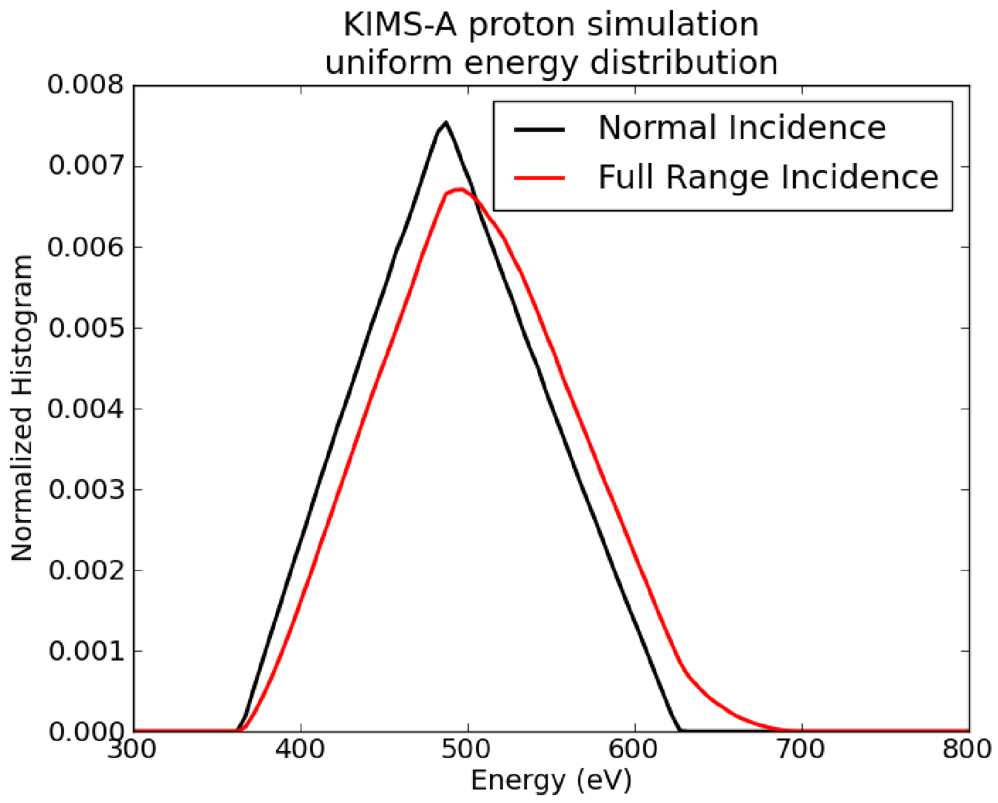


Figure 2.5: Response of one of the anode segments at normal incidence only, and with the full range (-6° to +19°) of azimuthal incidence. The triangular shape at a given incidence results from the mapping of the aperture width to the same width at the detector plane.

based on previous designs done at the University of Maryland and used spare components from those instruments. These earlier designs were also intended for cusp ion measurements. The basic parameters for the earlier instruments matched well with the needs of the RENU project. Changes to the heritage design were necessary to accommodate a new MCP detector system and a new telemetry interface.

2.3.1 Goals for KIMS

When KIMS was added to the instrument suite on RENU, a low-energy ion detector (observing ions of thermal energy to 800 eV) had already been included on the payload. The primary goal for KIMS, then, was to observe high-energy ions in the 1–10 keV range. The overlap of the lower energy range of KIMS with the other ion instrument on the payload provided the added benefit of redundancy and cross calibration between ion detectors.

KIMS is designed to take measurements at a rate of 250 Hz, determined primarily by the capabilities of the telemetry unit included on RENU. This sample rate is fast enough to potentially observe the cyclotron motion of ionospheric atomic oxygen. (O^+ in the Earth's magnetic field has a cyclotron frequency of about 100 Hz.) The ability to observe this motion would have the added benefit to KIMS of allowing a coarse mass discrimination not possible using only the momentum separation via cyclotron motion in the instrument's magnetic fields. Such a mass discrimination would be of interest, but not essential for the goals of the KIMS instrument.

2.3.2 Changes From Heritage

The original instrument design called for use of a set of 20 discrete channel electron multipliers (CEMs), separated into groups of 10 channels for each energy range (100–1000 eV and 1–10 keV). While robust and reliable, the CEM detectors proved to be prohibitively expensive with a long lead-time for manufacture. The KIMS design instead uses an MCP stack with a segmented anode as the detector. The anode design uses 10 pads for each of the energy ranges, keeping a similar configuration to the original CEM design. MCPs have been used in a number of sounding rocket flights, and various shapes and sizes are available commercially, off the shelf. However, the design for KIMS was constrained by the volume in which the original CEM detectors were to fit. As a result, a custom size MCP plate and holder were designed, manufactured, and qualified for robustness on a sounding rocket flight.

To accompany the MCP design, the instrument electronics were also redesigned. The new electronics make use of an inexpensive IC amplifier (as opposed to the more costly AmpTek A111 discrete amplifiers). The circuit was designed to operate at a high frequency, and the overall cost for the complete amplifier circuitry for all 20 channels was less than the price of one of the original amplifiers. The new design was tested and proved satisfactory for the anticipated flux rates of 10^4–10^5 per ($\mathrm{cm}^2 \cdot \mathrm{s} \cdot \mathrm{sr} \cdot \mathrm{eV}$) as observed by one of the earlier instruments on which the KIMS design is based (see Fig. 2.6). While ideal on a sounding rocket flight, the amplifier design is not radiation hardened, and would require further modification for orbital or exploratory satellites.

The MCP detector allowed the use of a lighter, lower power high-voltage power supply because of its low power consumption in operation. The high voltage circuit is designed with two supplies for redundancy, and still weighs less than the one supply design used in the earlier instruments. The design of the circuit allows changes to the applied voltage on the MCPs, requiring the change of only one or two resistors.

2.3.3 Basic Parameters for KIMS

The KIMS design uses two fields of magnitude 0.1 T and 0.27 T. These fields give energy ranges (for protons) of approximately 100–1000 eV and 1–10 keV over the length of the detection area. The detection area (or the size of the MCPs) is roughly the size of a standard microscope slide (2.5 cm x 7.6 cm). An aperture 3 mm wide and 17 mm high opens each region to a field of view that spans -6° to $+19^\circ$ in azimuth and $\pm 5^\circ$ in elevation (see Fig. 2.2).

An anode pad pattern of quadratically increasing pad widths was chosen to provide a similar $\Delta E/E$ response over the full energy range to that of the original CEM design (see Fig. 3.4). The chosen pad widths were constrained by the size of the detector itself, and so a constant $\Delta E/E$, requiring a much larger detector size for reasonable pad widths, was not feasible in this design. Each anode pad is connected to an individual amplifier and discriminator circuit with an output to an FPGA that counts the number of detected pulses in every 4 ms interval. The FPGA reports the count values of each anode pad to the rocket telemetry system at a rate

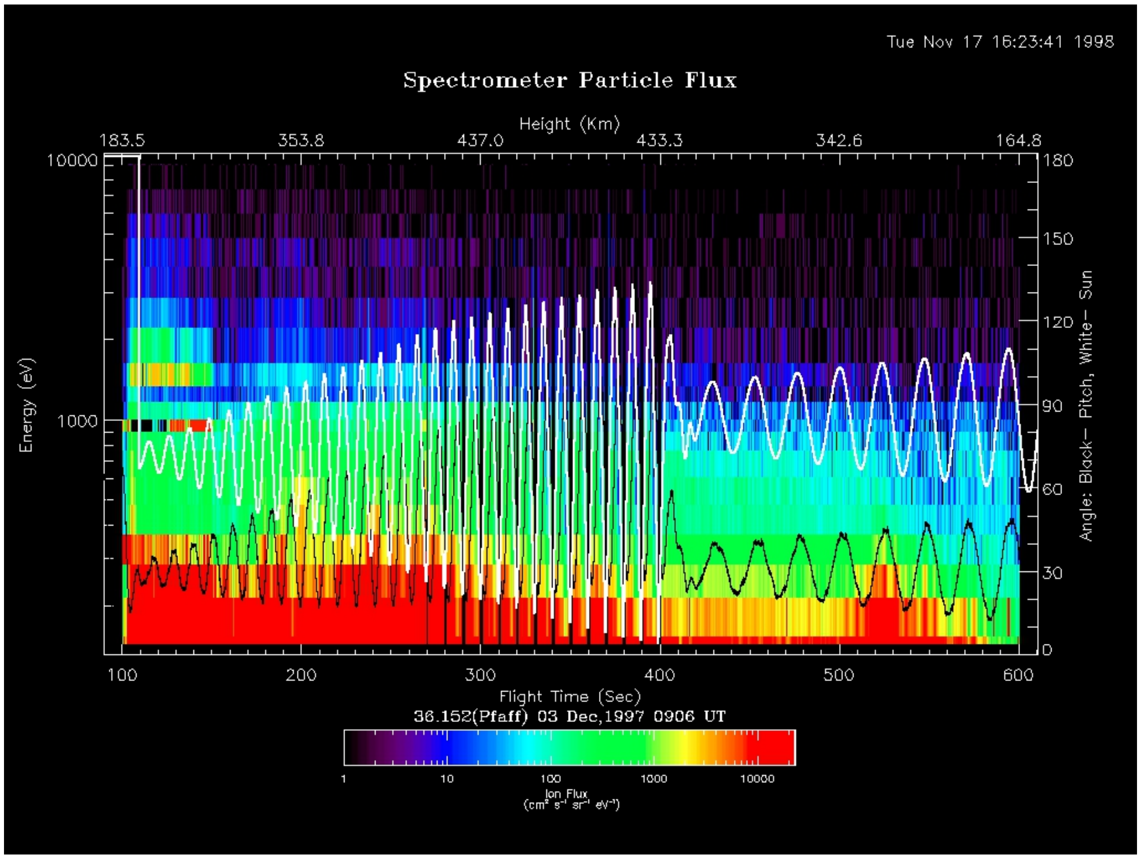


Figure 2.6: Figure showing results from measurements taken with instrument designs on which the KIMS instrument was based. The overlayed curves describe the rocket orientation applicable to this previous study. The contour plot indicates the flux values one can expect in the polar cusps. For KIMS, with a subtended solid angle of approximately 8.55×10^{-3} sr, a ΔE of about 200 eV (ΔE/E of about 0.25), and an anode size of roughly 1 cm^2 indicates an expected maximum count rate of roughly 10^4 counts per second per anode at low energies. The KIMS electronics are designed to be able to handle up to 10^6 counts per second.

of 250 Hz.

Power is provided to the instrument from the +28 V battery on the rocket. This voltage is regulated to 12 V for the high voltage power supply control and 3.3 V for the amplifier, discriminator, and FPGA telemetry circuitry. Two high voltage power supplies connected in parallel provide feedback-controlled -1.6 kV to the MCP stack. Total power consumption for the instrument during operation is estimated at a maximum of 8.4 W. Data from integration testing indicated the instrument performed at half this power on the ground when the MCPs were not fully powered. Full power consumption was estimated generously to provide a comfortable margin of error for flight conditions.

The instrument and electronics are enclosed in a μ-metal case with dimensions 9 cm x 17 cm x 20 cm. The combined yoke of the magnetic circuit and this enclosure are effective in shielding the strong magnetic fields inside the instrument, and reduce the stray field to 10 nT at a distance of 1 m, as tested at the Magnetic Calibration Facility at NASA's Goddard Space Flight Center (see Sec. 3.3.1). An aluminum bracket was designed to mount the instrument vertically on the rocket payload, with the azimuthal view along the rocket's field-aligned trajectory. The instrument view is centered at about $+6^\circ$ above a directly radial view to the magnetic field. The entire assembly weighs less than 4.5 kg.

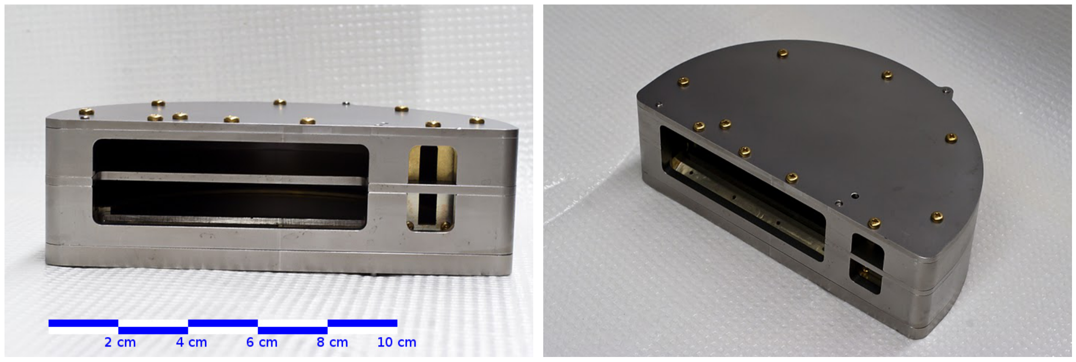


Figure 2.7: Photos of the magnetic circuit. The apertures are the smaller openings on the right side. The partition can be seen between the two halves through the space where the detector is mounted on the left side. The permanent magnets are attached to the interior of the top and bottom plates, secured with aluminum clamps. These mangets create fields of 0.1 T on the top and 0.27 T on the bottom.

2.4 KIMS Design

2.4.1 Magnetic Circuit Design

The magnetic circuit refers to the portion of the instrument that creates the constant magnetic fields needed to accurately disperse the incoming ions by their momentum per charge. The magnetic circuit is an assembly of specially designed pieces (the yoke) around two permanent magnets. The yoke provides a return path for the magnetic field and shields the exterior of the instrument from the strong magnetic fields inside. The two permanent magnets, of different strengths, are mounted inside the yoke to create the magnetic field.

The hallmark of the magnetic circuit design is the Vanadium Permendur[2] yoke that encases the magnets. Vanadium Permendur is a steel alloy with a high cobalt concentration and a small portion of vanadium. The alloy is heat treated

[2] Vanadium Permendur is also called Hiperco 50™

carefully before machining, giving it a high magnetic permeability that does not saturate in strong magnetic fields. The thickness of each Vanadium Permendur part is determined through careful simulation of the magnetic fields to optimize the homogeneity of the field strength inside the instrument throughout the detection region. Vanadium Permendur is brittle and its magnetic properties sensitive to both heat and shock. All of the parts made from this alloy were machined via wire EDM.

The magnetic circuit assembly consists of two halves, with one magnet for each side of the instrument. Each magnet is a semi-circular plate with a 6 cm radius. Each is fabricated with an assembly of custom designed segments to create a uniform field across the face of the plate. The magnets in our design have differing strengths of approximately 0.1 T and 0.27 T. Each magnet is placed carefully on one of two Vanadium Permendur plates that become the top and bottom of the magnetic circuit. The magnets alone are strong enough to keep themselves in place, however aluminum clamps around the magnet pole faces prevent them from shifting during vibration. Each magnet is then surrounded with a Vanadium Permendur shell. Openings in the shells provide for the instrument apertures and the detector assembly. The two sides of the magnetic circuit are brought together with a Vanadium Permendur partition that separates the two regions of magnetic field. The entire assembly is aligned using three non-magnetic, stainless steel roll pins. Brass screws are used to secure the clamps in place.

The final part in the magnetic circuit is a wire mesh grid placed at the face of the detection plane opening. This grid is grounded along with the entire magnetic

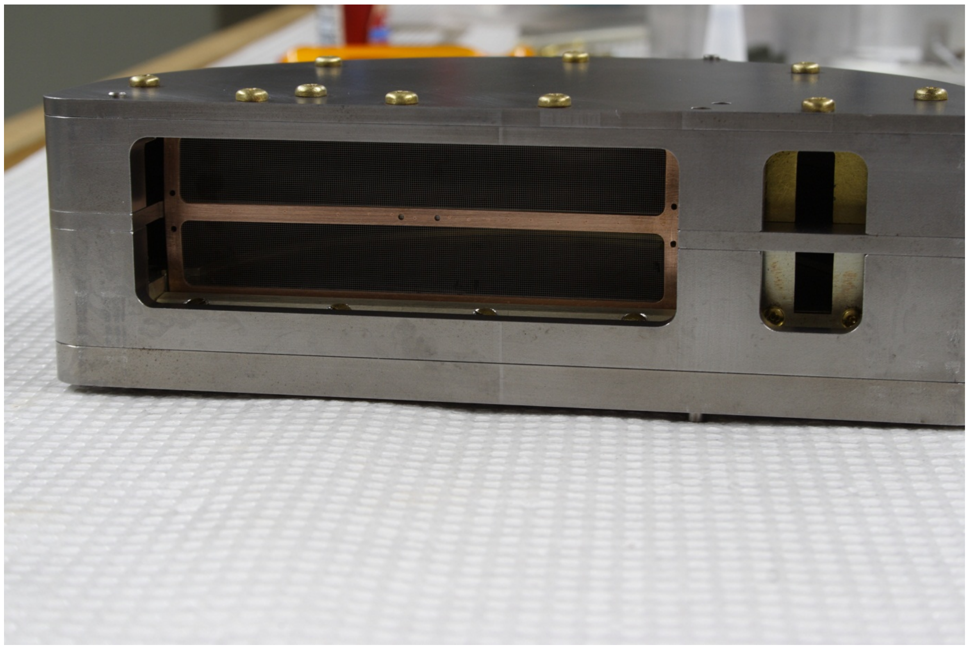


Figure 2.8: Photo showing the mesh grid in place on the magnetic circuit. An aluminum clamp holds the grid in place. The entrance apertures for the two sides of the instrument are visible on the right.

circuit to prevent the electric fields of the detector assembly from distorting the cyclotron motion of ions inside the instrument. The grid is 70 lpi, 90% transmission, nickel mesh sandwiched and spot welded between two phosphor bronze frames.

2.4.2 Mechanical Design

The magnetic circuit is held in a close fitting, aluminum bracket referred to as the clamshell. The clamshell halves bolt together, securely holding the magnetic circuit while eliminating the need for fasteners in the brittle Vanadium Permendur parts. The clamshell also serves as a mounting platform for all of the electronics and the detector assembly. The entrance aperture on the clamshell is cut to provide the -6° to $+19^\circ$ azimuthal field of view into the instrument.

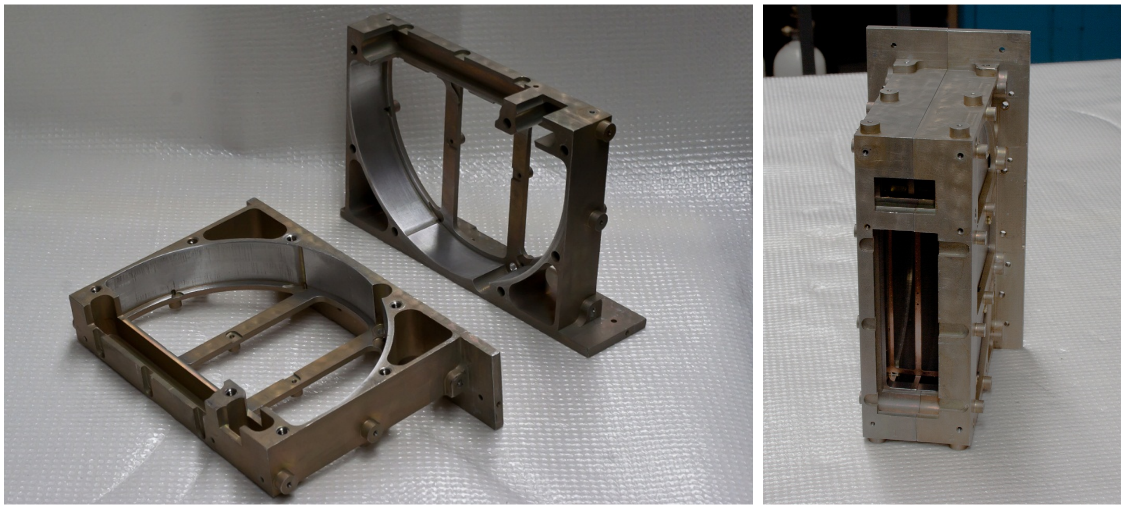


Figure 2.9: The clamshell halves and assembled around the magnetic circuit. The detector and electronics are all fastened to the clamshell using tapped bosses on the outside surfaces.

The circuit boards are mounted on the instrument sides with threaded standoffs. The standoffs also support the high-permeability, μ-metal shield that covers the entire instrument. The shield has openings for the two apertures, the three cable connectors for power and telemetry, and a safety plug that prevents the high voltage from being activated accidentally while at atmospheric pressure. All of the fasteners and connectors in the instrument are made of non-magnetic materials.

A mounting bracket designed specifically for the RENU sounding rocket was fitted for the KIMS instrument. The bracket fastened into the bottom and the back of the instrument while leaving access to the cable connections. The bracket is bolted to the rear bulkhead of the payload, giving the KIMS instrument a view radial to the axis of the payload.

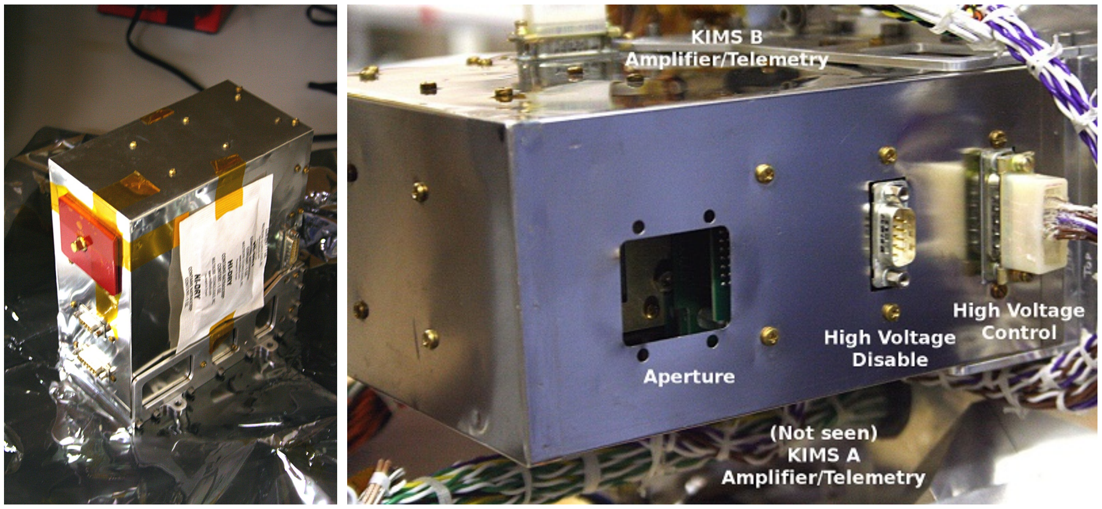


Figure 2.10: The KIMS instrument enclosed in its μ-metal case and attached to its mounting bracket, and mounted on the RENU main payload. The red plate in the first image covers the aperture to the instrument for shipping.

2.4.3 Detector Design

The MCP detector stack was designed to fit in the space originally allotted for the CEM detector array. An MCP stack is typically held by compression in a housing made of an insulating material. The compressive housing, by its nature, masks a small portion of the detection area of an MCP. The KIMS detector thus has a 24% reduction in detector area in comparison to the CEM array due to the housing, however the difference is negligible for the anticipated ion fluxes in the cusp.

2.4.3.1 MCP Housing

The MCP housing is built from four pieces: an outer housing, an inner housing, and two “rabbit ears” that extend to the position of the high voltage supply board. The rabbit ears are fixed to the inner housing, but the inner and outer housings are

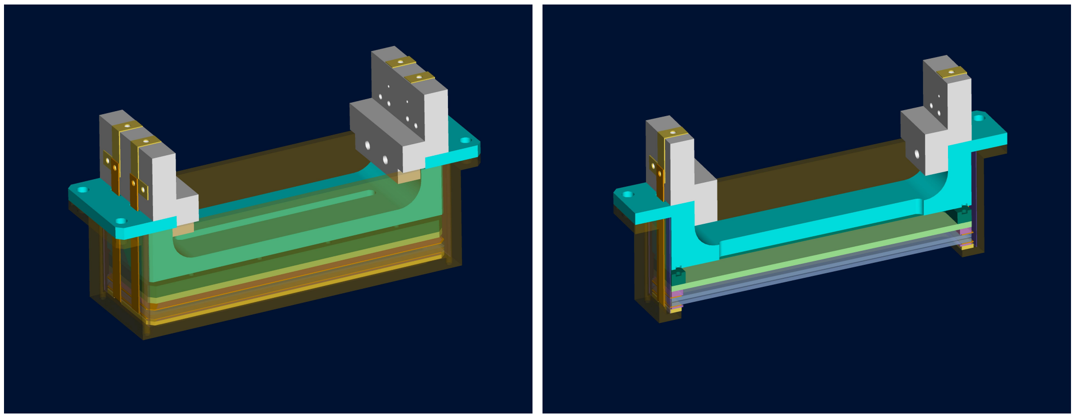


Figure 2.11: Engineering drawing and cross-section of the KIMS MCP detector. The individual components seen in the cross-section are, from bottom to top: outer housing, HV grid, insulator, contact ring, MCP 1, contact ring, MCP 2, contact ring, insulator, anode board, spring plate, and inner housing.

maintained in place when connected directly to the instrument. This arrangement eliminates excess connections, minimizing the size and weight of the detector.

2.4.3.2 MCP Stack

The MCP stack assembly consists of (in order) a grid frame held at a high negative voltage, an insulator, an electrical contact, the first MCP, an electrical contact, the second MCP, an electrical contact, an insulator, the anode board, and finally the spring plate. The spring plate and inner housing have matching blind holes that accommodate a set of ten small springs. This mechanism provides adequate compression for maintaining electrical contact to the plates while protecting the plates from vibration during launch.

The electrical contacts, including the front grid, are rectangular frames with extended tabs bent at 90°. These tabs are fastened to a set of conductive tabs on the rabbit ears. The rabbit ear contacts are bent over the top of the rabbit ears, where they are fastened to the high voltage board. This configuration provides electrical connection between the electrical contacts in the stack to the high voltage board. To prevent shorts between the extended tabs and the frames within the stack, a pair of thin Ultem^{TM} inserts on either side of the stack slide between the tabs and the stack pieces.

The anode board is a gold plated, etched PCB with a set of 22 pins extending from the rear. Each region of the magnetic field is exposed to one half of the board. The ten pads on each half are sized to produce reasonable $\Delta E/E$ values over the full range of energies while maintaining safe pad spacings to minimize cross-talk between the pads. The pins, a set of 22 0.1" pitch square header pins, connect to the first board in the electrical stack assembly to send the charge pulses of each pad to their respective amplifiers. Two of the pins provide a ground connection for ground planes above and around the anode pads.

The detector assembly is fastened to a plate that mounts on the face of the instrument. When in place, the front of the detector is sufficiently offset from the grounded grid inside the magnetic circuit to prevent arcing when the negative high voltage is applied to the MCP stack. This mounting method also allows easy access and removal for assembly and testing of the instrument.

2.4.4 Electrical Design

The electrical system consists of two primary components: the two amplifier/telemetry boards, which have the electronics for counting, processing, and for-

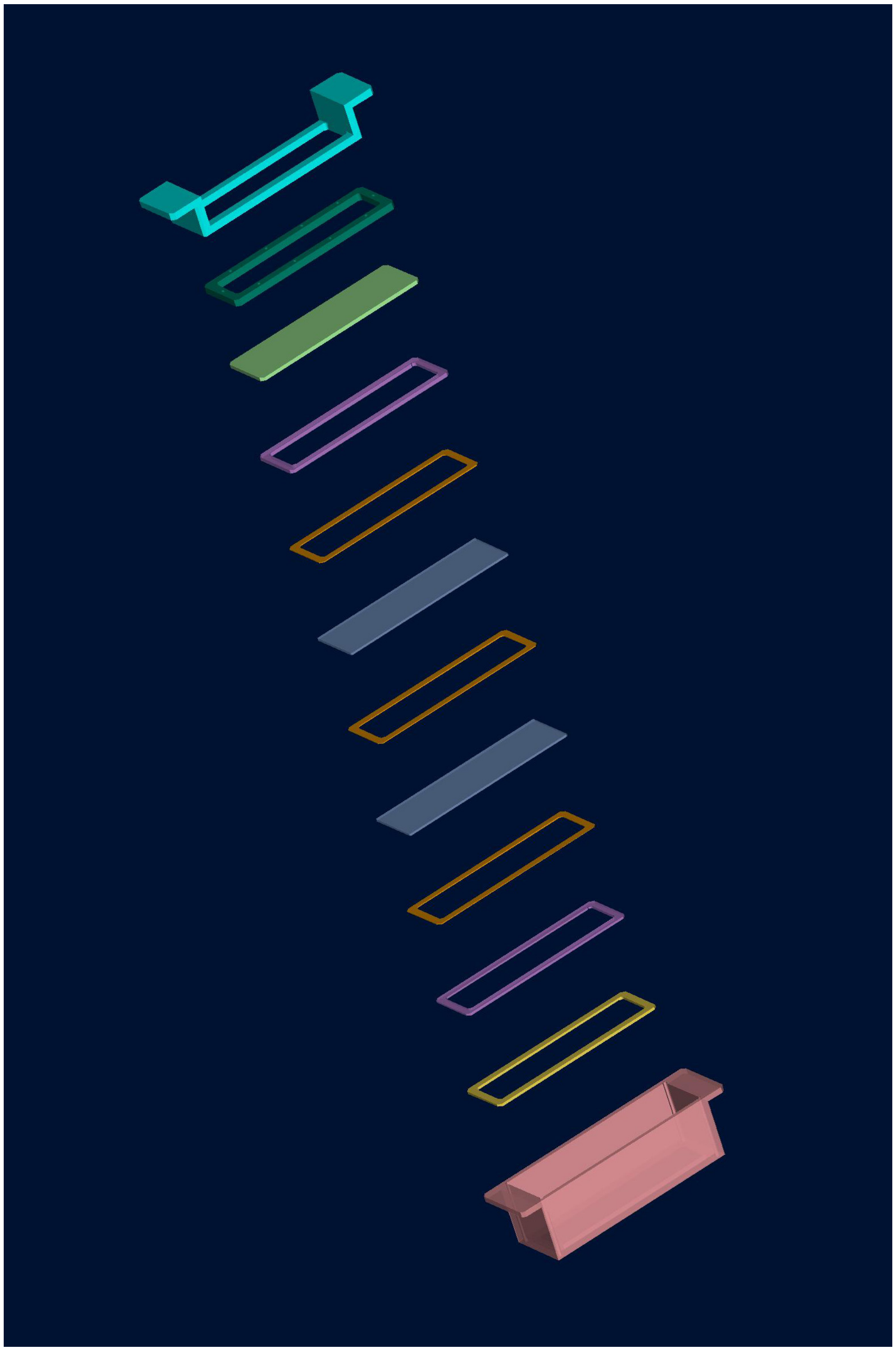


Figure 2.12: Exploded view of the MCP stack as a rough sketch for all the parts. From bottom to top are the outer housing, the grid frame, an insulating ring, electrical contact, the first MCP, electrical contact, second MCP, electrical contact, insulator, the anode board, the spring plate, and the inner housing. Not shown in this sketch are the tabs on the electrical contacts or the ears on the inner housing.

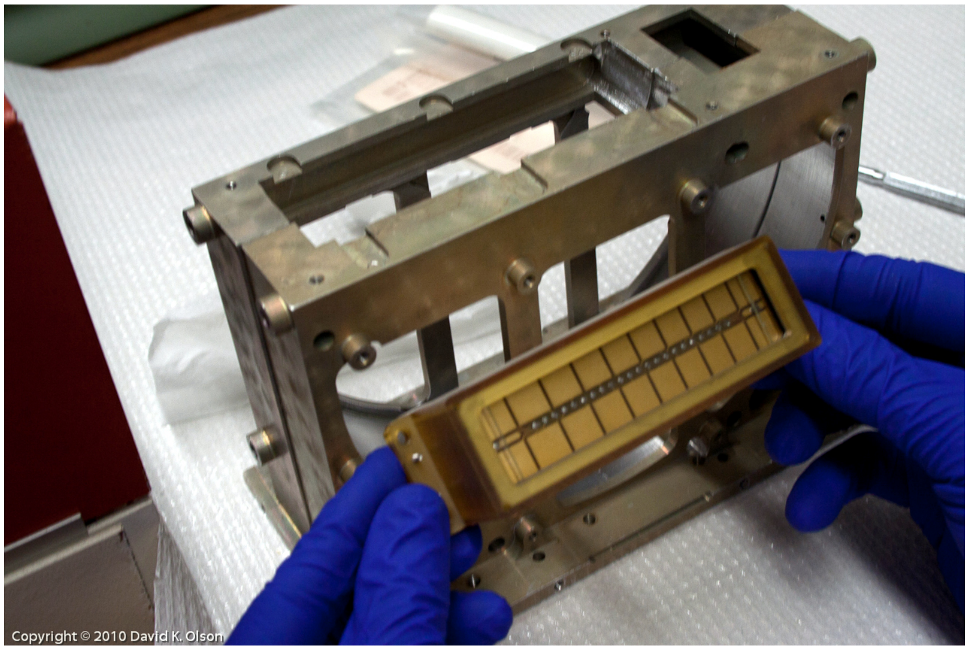


Figure 2.13: View of the anode board inside the detector. Glass slides are used here in place of the MCPs for mechanical testing to prevent contamination of the MCPs.

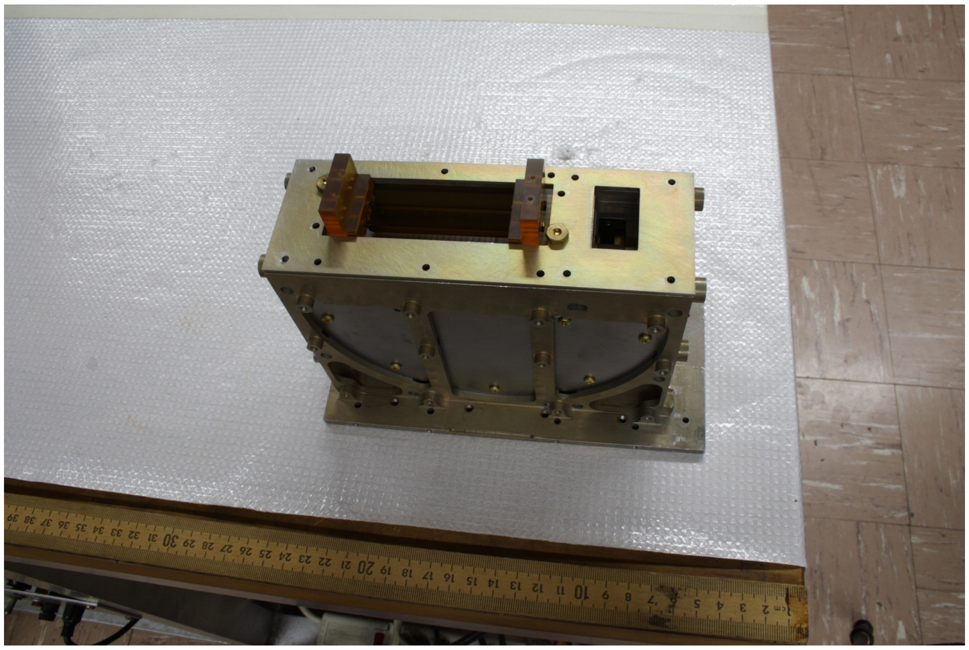


Figure 2.14: Mechanical fit check of the MCP stack design. The front end of the detector is offset from the grid inside the magnetic circuit to prevent arcing.

matting for transmission during flight; and the electrical stack, which provides power to the MCP stack. The two amplifier boards attach to the sides of the clam shell, while the electrical stack assembly attaches to the front of the clam shell, directly above the detector. The stack assembly itself consists of three parts: a distribution board, which connects the anode board pads to the two amplifier boards; a ground plane, which shields the pulse signals from noise; and the high voltage supply and control boards, which turn the high-voltage power supplies on at the appropriate time during flight.

2.4.4.1 Amplifiers and Telemetry

The amplifier portion of the detector system consists of a charge sensitive pre-amplifier followed by a shaping amplifier. These functions are both provided by an individual OPA 2354 dual op-amp chip for each pad on the anode board in the detector. Each amplifier output is sent to an individual discriminator, provided by an LMV331 comparator, to detect pulses. The input of the pre-amplifier is also connected to a 1 pF test input for calibration purposes (see Appendix A.4). Each of the two amplifier boards has a collection of 10 amplifier-discriminator pairs, one for each pad on the corresponding side of the instrument.

Calibrating the amplitude of the shaped pulse from a known charge input allowed us to determine the voltage threshold for detected pulses. Pulses with a minimum of 500,000 electrons are counted. The pulse discrimination threshold is typically set by examining the dark count pulse height distribution of the MCPs

[29], but this property was not examined for the KIMS instrument MCPs due to time and equipment constraints. Instead, a threshold typical to the types of MCPs used in the instrument was chosen. The gains of the MCPs used are high enough that there are typically more than 10^6 electrons in an event pulse from a pair of plates, so 5×10^5 electrons is a reasonable threshold while keeping above typical noise levels.

Each discriminator provides a logic pulse output that is connected to one of 10 inputs of an FPGA. The FPGA is programmed to count pulses over a 4 ms period. The counts for each channel are buffered, formatted, and sent to the main payload's pulse code modulation (PCM) stack to be transmitted down to the tracking station. All formatting for the KIMS instrument data is done on-board, so a simple serial data stream of the values in each amplifier's counter in the FPGA is sent to PCM stack of the rocket.

2.4.4.2 Electrical Stack Assembly

The electrical stack assembly consists of four circuit boards stacked together and attached to the detector stack. These boards distribute the anode signals to the two amplifier/telemetry boards and control the high voltage applied to the detector.

The distribution board has traces that connect the anodes in the anode board to the individual amplifiers. The traces on the anode board lead to a pin configuration such that we have a pattern of pads A0, B0, A1, B1, ... , A9, B9. This designation uses A and B to distinguish between the two magnetic field regions,

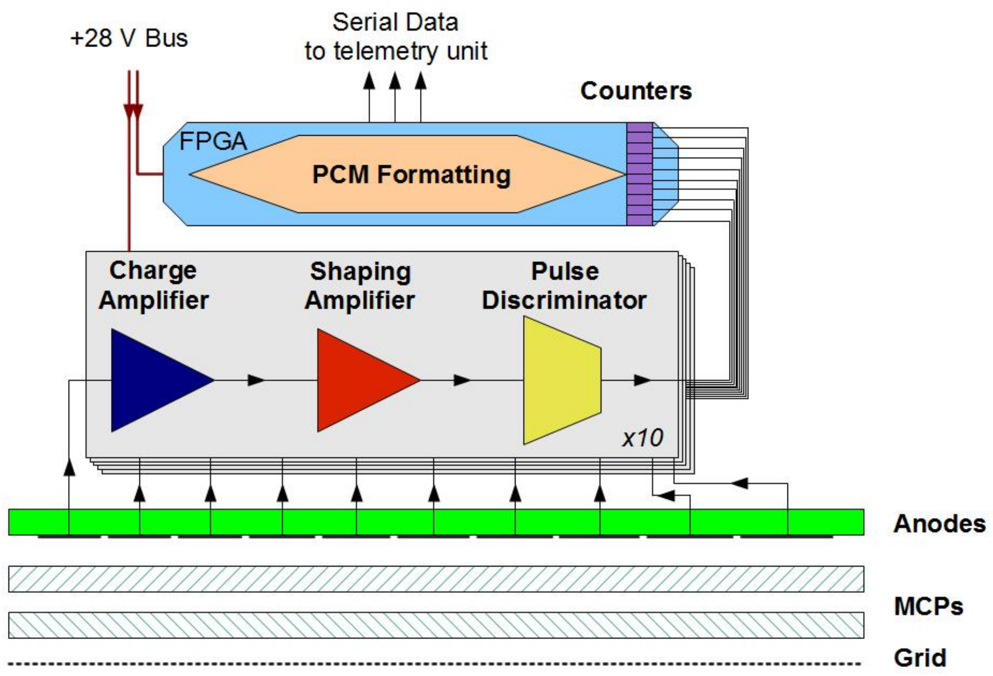


Figure 2.15: Diagram showing the detector signal processing. There are 10 amplifier units for each side of the instrument. The 10 units are counted individually in an FPGA that formats the data for the telemetry stream.

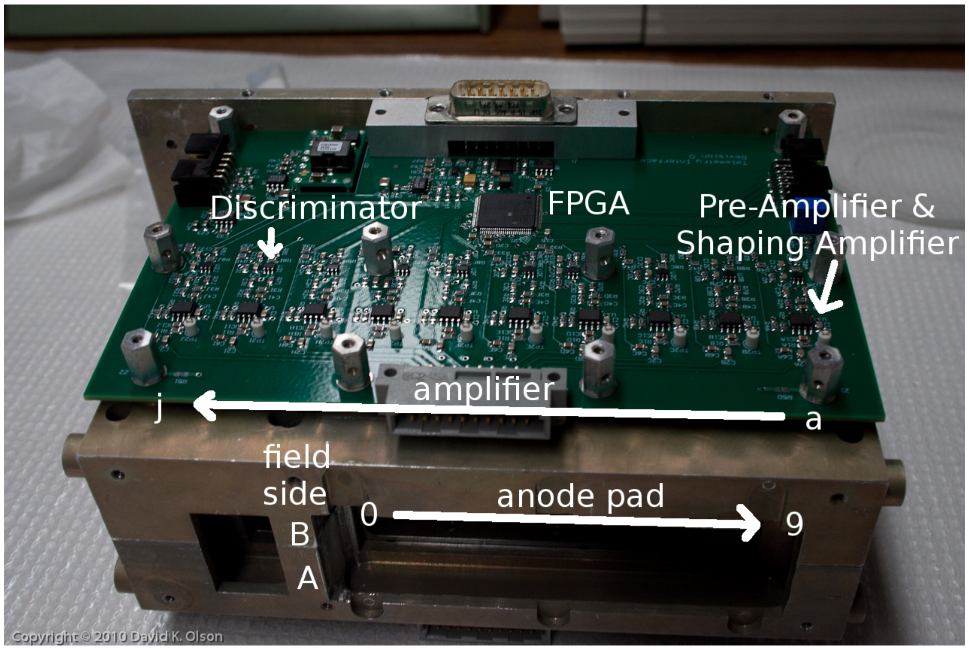


Figure 2.16: Amplifier/Telemetry board mounted on the KIMS instrument. An identical board is mounted on the opposite side. Each amplifier (a–j) consists of a pre-amplifier and shaping amplifier in a dual op-amp package. The amplifiers are followed by a discriminator. The outputs of each of the 10 discriminators are connected to individual channels in the FPGA and counted. The FPGA reports the counts from each 4 ms interval to the payload telemetry unit. Note that because the boards are identical, amplifier a is connected to anode pad 0 on the KIMS-A low energy side, but to anode pad 9 on the KIMS-B high energy side. The telemetry is formatted in the amplifier order A-a, B-a, A-b, B-b, etc., or equivalently to anodes A0, B9, A1, B8, etc.

with A corresponding to the 0.1 T side and B to the 0.27 T side. The numbers 0-9 identify the individual pads in order of increasing energy. The pad widths increase from pad 0 to pad 9. The amplifiers on each board are labeled a through j. Since the amplifier boards are identical, the amplifier sequence reverses between the two sides due to the amplifier board orientation on the instrument; A0-9 connect to amplifiers a–j respectively on their amplifier board, while B0-9 connect to amplifiers j–a respectively on their board (see Fig. 2.16).

The ground plane is a two-sided PCB with no traces. The soldermask is removed in the areas where the assembly is fastened together to ground the two sides of the board to the instrument case. This piece is simply to isolate and minimize noise in the MCP anodes.

The high voltage supply board consists of two independently controlled Emco Q30-12 high voltage DC–DC converters. These power supplies have a proportional output controlled by a 0–12 V input. They are connected in parallel to provide redundancy during flight. The rabbit ears from the detector stack extend beyond the distribution board and ground plane to contact the pads on this board that provide the voltages at the various points. These are fastened with 0-80 screws to secure the electrical connection. The high voltage supply is connected across a voltage divider, using the resistance of the MCPs in series with high-voltage resistors on the supply board. The supplies are set to provide -1.6 kV across the entire detector stack, with an approximately 70 V drop from the front grid (at -1.6 kV) to the first MCP, as well as the second MCP to the anode board (at ground). This voltage was determined to be within the optimal operating range for the MCPs used in the detector.

Finally, the high voltage control board that drives the HV supplies connects to the supply board. A feedback loop on this board is used to maintain a constant -1.6 kV output from each supply. In addition to controlling the high voltage supplies, this board also connects to the PCM stack to report three analog values in each 4 ms time interval. The feedback voltage on each high voltage supply is reported, as is the temperature of the control board.

2.4.5 Ground Support Equipment

Ground support electronics (GSE) were developed to accompany the instrument during testing and integration. The GSE system provides a simulated connection to the rocket's telemetry unit to check the condition of the instrument and obtain calibration data. The GSE uses an FPGA to simulate the telemetry communication and a TI MSP430 microcontroller to interface the instrument to a computer via USB. The GSE was used for both testing and calibration of the KIMS instrument.

The amplifier boards are programmed to provide a test signal into each amplifier. Pulses at a specified frequency are sent from the FPGA to the test input of each amplifier on the board, using progressively lower frequencies on each successive amplifier (dividing by a factor of 2 at each amplifier). This test signal provides a known count-rate pattern that is reported back to verify that the circuit is working properly. Each amplifier/telemetry board is programmed with a unique base

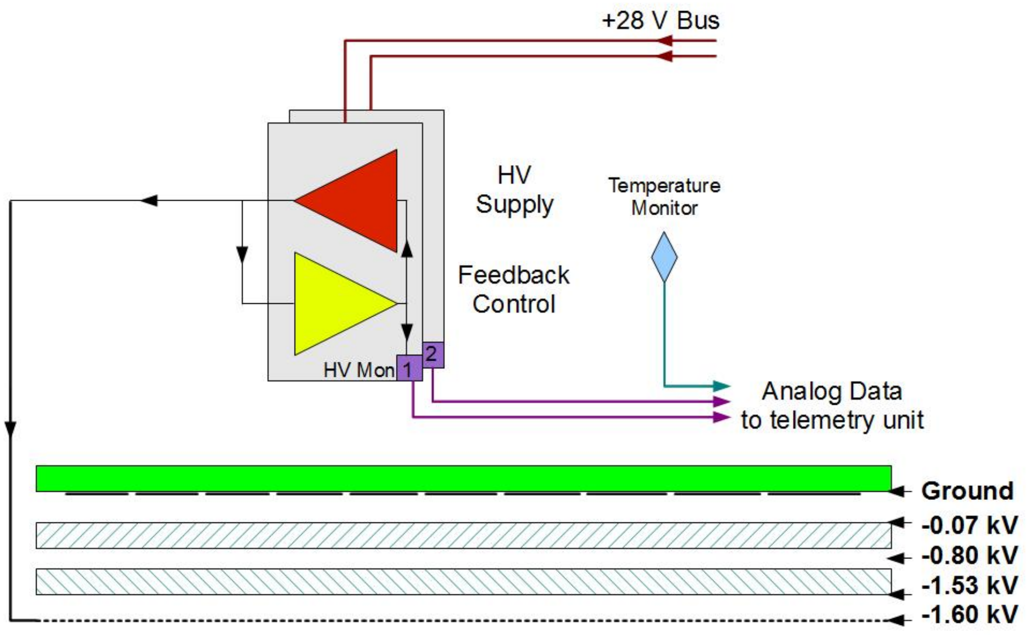


Figure 2.17: Diagram showing the high voltage system. A feedback loop from the high voltage supply output is used to keep the output steady through flight, even if the bus voltage were to change. Three analog signals monitoring the health of the power supplies are measured and included in the telemetry. These measure the temperature and voltage values of the high voltage control board.

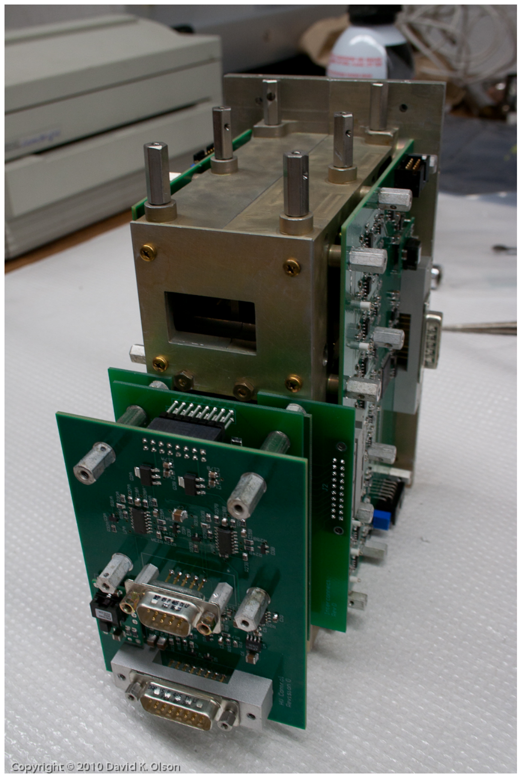


Figure 2.18: Electrical Stack Assembly mounted on the KIMS instrument. The top board seen here is the high voltage control board. The high voltage power supplies are on the board immediately behind it, followed by the ground plane and distribution board that connects to the amplifier boards on either side.

frequency to distinguish the signals between the two sides of the instrument, using 100 kHz on the KIMS-A board (each subsequent frequency is reduced by a factor of two for the 10 amplifiers on the board) and 200 kHz on the KIMS-B board. The GSE is programmed to issue the same commands to the boards for testing purposes.

For testing and calibration, the GSE uses an MSP430 microcontroller to interface an FPGA simulating the rocket PCM stack with a USB serial connection to a computer. The MSP430 programming is a simple system that triggers from the simulated telemetry signals given by the GSE FPGA. When the FPGA simulates a frame-ready state of the PCM, the MSP430 reads out the proper data from the telemetry frames corresponding to KIMS data, formats the values into 16-bit hexadecimal, and reports the values via serial USB to the computer. A serial terminal program capable of logging the data stream to a file is used to record the data. The instrument itself continually takes and reports data, but data are only recorded when requested. For testing purposes, the MSP430 was programmed to wait for a single character command input from the terminal program and then record the next 100 samples reported by the instrument telemetry boards.

2.5 KIMS Summary

In summary, the KIMS instrument uses an efficient permanent-magnet system to disperse ions by their momentum. For a given ion mass, this dispersion is equivalent to a separation by energy. Each side of the instrument measures ten ranges of energy simultaneously, with side A measuring 100 to 1000 eV, and side

B measuring 1 to 10 keV. An MCP detector is used to count individual ions incident to the detector plane, with 10 anodes on each side collecting the charge pulses from the MCP stack. The charge pulses are counted with an inexpensive integrated circuit amplifier/discriminator pair and formatted for telemetry by an FPGA. The MCP stack is powered with a high-voltage DC–DC converter using feedback control to maintain a constant bias voltage to the MCPs through the course of flight. The entire instrument is enclosed and shielded to minimize stray magnetic fields, and provides an excellent data collection rate at a size and mass compatible with sounding rocket requirements.

Chapter 3

KIMS Analysis, Data, and Results

The KIMS instrument was thoroughly tested and calibrated prior to flight. Instrument properties were also modeled numerically, and the tests performed show excellent agreement with the model calculations. The instrument electronics were also tested and adjusted to optimize operating parameters for the particular sounding rocket flight. In addition, the instrument underwent vibration and magnetic testing to qualify for the rocket flight profile and parameters. The magnetic testing also provides understanding of what impact stray magnetic fields from KIMS might have on other instruments on the payload.

Operational testing done at the University of Maryland used an established vacuum chamber system. MCPs require pressures below 2×10^{-6} torr for safe operation. Calibration testing was done at a pressure of approximately 1×10^{-6} torr. An electron bombardment ion source provided the ions for the tests in this chamber (see Fig. 3.1). This type of ion source uses a hot tungsten filament as a source of electrons. These electrons are accelerated into a chamber with the gas to be ionized. The ions are then accelerated from the chamber, passsing through an electrostatic lens to form a focused beam of ions. A positive voltage applied to the ionization chamber is used to accelerate the ions. Because the ions are created with a very low directed velocity, this voltage also determines the energy of the ions in the beam

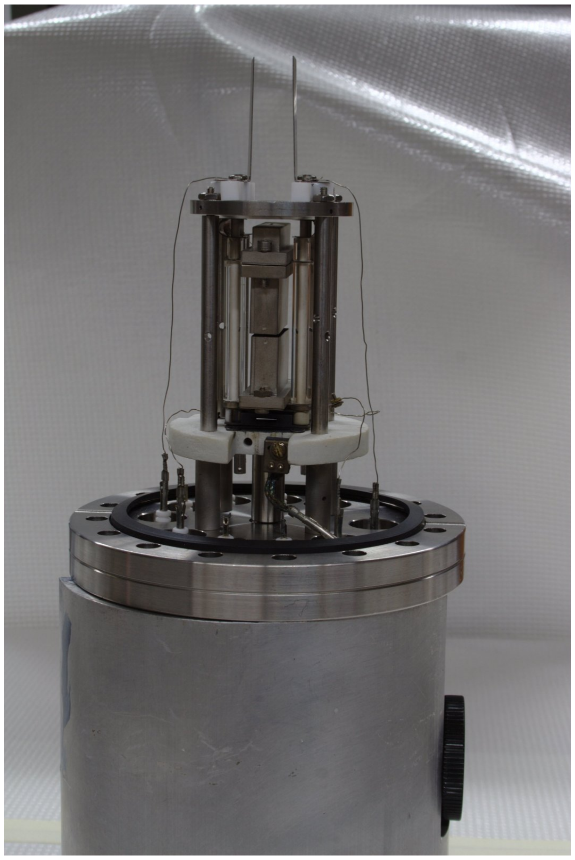


Figure 3.1: The ion source used for KIMS testing and calibration. The test gas is leaked in through the flange seen at the bottom of the stack. Tungsten filaments fastened to the ceramic plate emit electrons that are accelerated into a collision chamber just above the ceramic plate. The energetic electrons ionize the gas in the collision chamber. The ions are extracted into the electrostatic lens above the collision chamber. The resulting ion beam can be deflected by applying a voltage to the two plates at the top of the source.

leaving the source. The design of the particular source being used could be operated safely to a maximum ion energy of 3 keV.

3.1 Modeling and Operating Parameters

3.1.1 Instrument Simulation

A Monte Carlo simulation of the KIMS instrument operation, written in the Python language, provided a benchmark for instrument performance. The magnetic fields in the instrument and the ion masses are fixed in the simulation. Simulated ions are given uniform distributions of energy over a specified range, entrance offset from the aperture center, and azimuth within the instrument field of view by random sampling over each parameter. Sufficient random samples are accumulated to scan the entire response of the instrument with adequate statistics. By calculating the position an ion strikes the MCP at the end of its trajectory, we can determine the energies, offsets, and angles that reach particular sections of the detector plane. These correspond to the expected response of a chosen anode design. The code for this simulation can be found in Appendix B.1.

As anticipated from the linear relationship of the detection position to the finite aperture width, the response functions for each anode take a nearly Gaussian shape as seen by the obtained histograms as seen in Figure 3.2. The histograms provide an expected energy distribution over each anode, and are characterized by the mean expected energy. The histograms also provide a width measurement ΔE given by taking the width of a rectangle having the same height and area as the histogram. By using normalized histograms, this width is simply given for each curve by $\Delta E = \frac{1}{h_{max}}$. A plot of the calculated mean energies and their corresponding ΔE values is given in Figure 3.3.

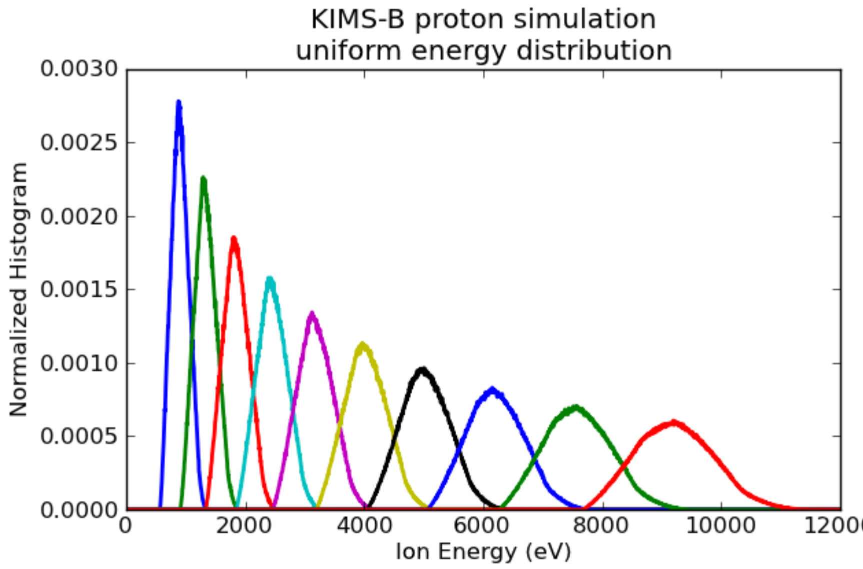
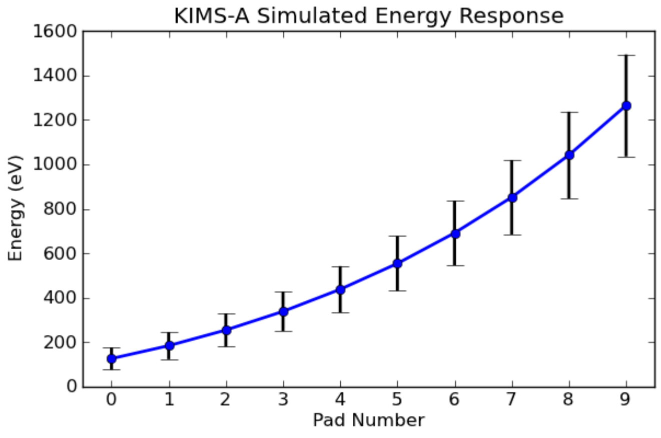


Figure 3.2: Histograms from a simulation for the high energy side of KIMS using a uniform energy distribution , offset distribution from aperture center, and entrance angle distribution. The histograms are normalized over the total counts in the simulation such that each curve has an area of 1. This set of curves provides a mean energy and energy spread ($\Delta E/E$) for each anode segment, with the curves from left to right corresponding to pads B0 through B9.



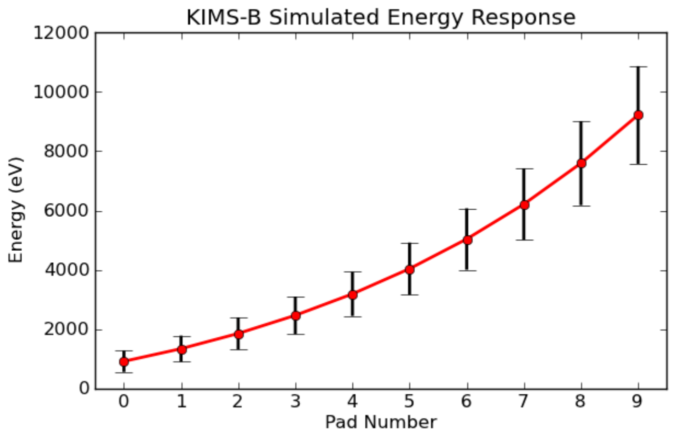


Figure 3.3: Simulation calculations showing the mean energy and expected ΔE for each anode. The mean energy is given by the location of each point and the value of ΔE by the vertical bar at each point on the plot.

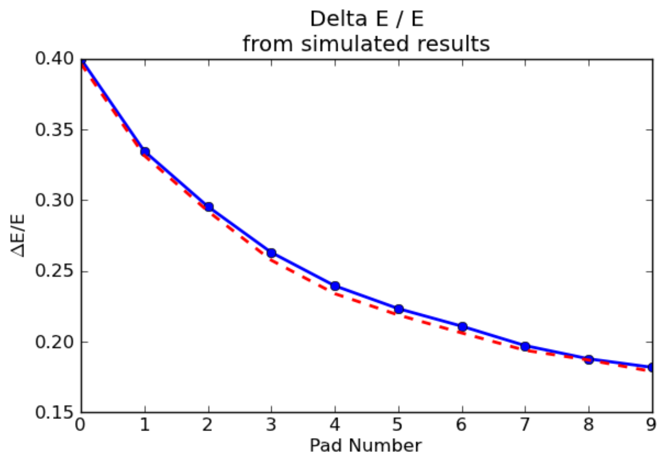
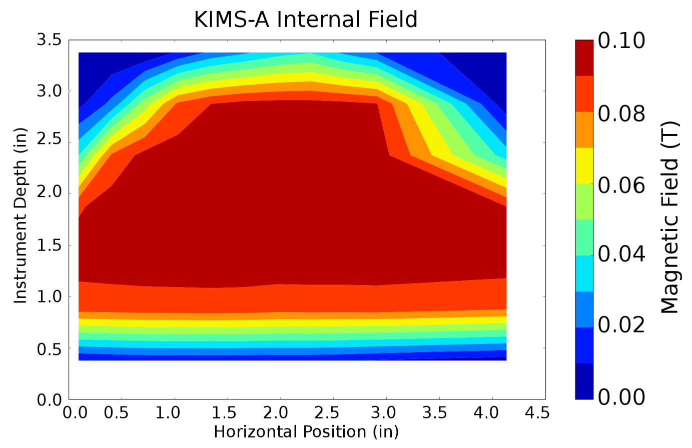


Figure 3.4: Plot of $\Delta E/E$ from the KIMS simulation. The dashed line corresponds to a best fit of a quadratic curve to the values. The anode pad dimensions were chosen to give a response similar to the original CEM design, but with a smoother change in $\Delta E/E$ over the array of anode segments.

3.1.2 Magnetic Field

The magnetic field of the instrument was measured in two different ways. The first used a gaussmeter with a Hall probe. A jig made from Delrin placed inside the opening for the detector allowed the Hall probe to be precisely located at half of the gap height on either side of the magnetic circuit. Field strengths were measured in the center of each side of the instrument at various depths into the magnetic circuit to produce the maps shown in Figure 3.5. A major issue with the gaussmeter is the lack of a precise calibration, though a comparison to a typical value for Earth's magnetic field suggested it was accurate to within 20%. An estimate of the Hall probe calibration provided a measurement of the low field at 0.098 T and the high field side at 0.278 T. This measurement may have significant absolute error, but it does show clearly the uniformity of the field in the interior of the KIMS instrument.



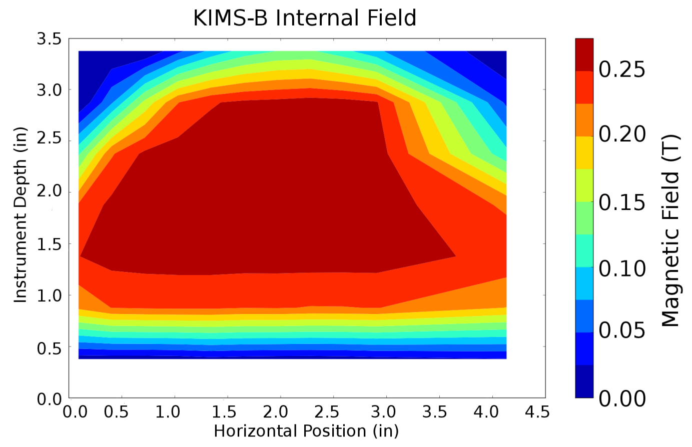


Figure 3.5: Measured fields on the interior of the KIMS instrument. A refers to the low energy side (with anodes A0-9), and B to the high energy side (with anodes B0-9).

The homogeneous field is expected for the magnetic circuit design. Preliminary measurement of the magnetic field as a function of height in each side of the magnetic circuit also showed uniformity within 10%.

The instrument was then placed in a vacuum chamber with a bare anode detector in place of the MCPs. An electrometer was used to detect the small currents collected on each anode. The ion source, using hydrogen gas, scanned the responses of each anode over ion energies from 50 to 1200 eV. The dominant signal corresponding to the H_2^+ base is clear, but background gas also provided weaker signals

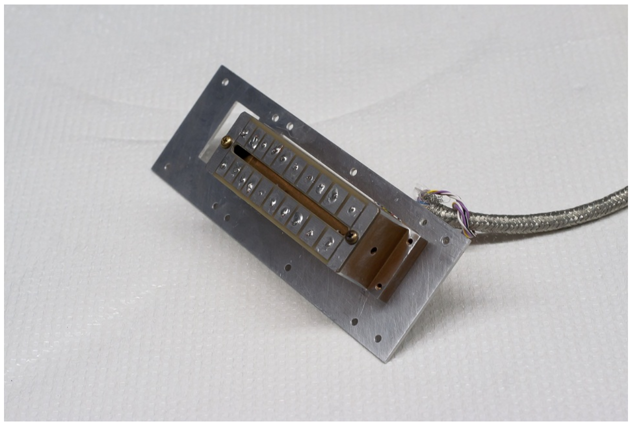


Figure 3.6: Photo of the bare anode detector used for measuring the absolute field strength. The conductive pads are exposed directly to the flux of ions at the detection plane in the instrument. The pads collect incident ions and the generated currents (on the order of 10's of pA) are measured by an electrometer. The detector is attached to a plate that fastens to the instrument's clam shell.

in water and nitrogen. A weak proton signal can be seen as well. The results of this test are shown in the contour plot of Fig. 3.7. Superimposed on this plot are the predicted positions of each ion species from the Monte Carlo model. Because the model predictions match these results well, we can conlude that the absolute values of the magnetic fields are, in fact, very close to the intended design values of 0.1 T and 0.27 T used in the model.

3.1.3 MCP Applied Voltage

MCPs require a bias voltage across their thickness to operate, and can be operated over a range of voltages. The optimal voltage for operation depends on the plate properties and resistances, and is unique to each set of plates. As a general rule of thumb, a nominal voltage of about 1 kV per plate is needed. To determine the optimum voltage for the plates used in KIMS, we examined the response of the MCP

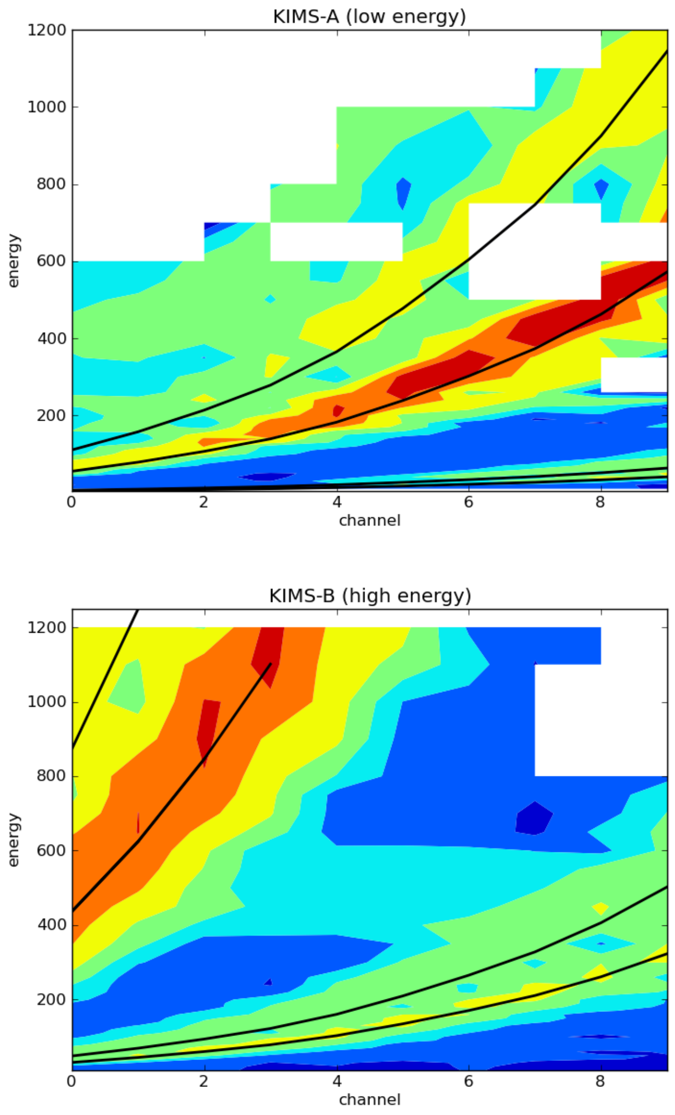


Figure 3.7: Bare anode results with simulated response lines overlayed for the two sides of the KIMS instrument. Simulation curves correspond to, from top to bottom, H^+, H_2^+, H_2O^+, and N_2^+. The contour plot shows the general response of the instrument in energy for anode pads 0–9 across the horizontal axis. The current at a given pad is a function of energy as previously discussed. The measured response using the bare anode matches the simulation quite well, with the difference being attributed to the uncalibrated values of the magnetic field measurement. This plot uses the estimated 0.098 and 0.278 T of each field. The low energy side suggests the estimate is low, and actually closer to the expected 0.1 T.

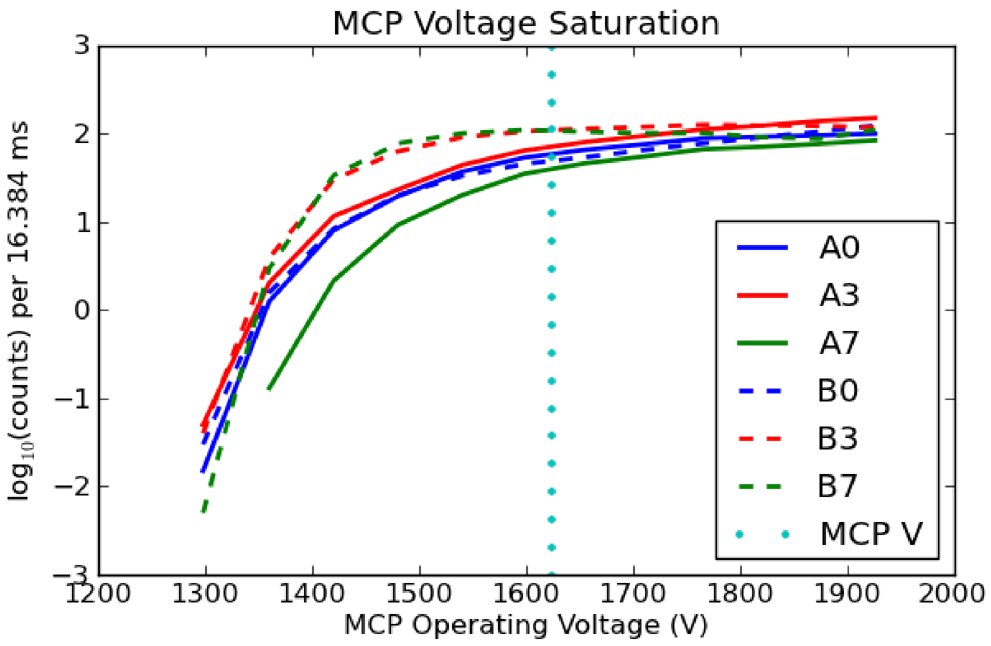


Figure 3.8: Plot showing the MCP response vs. applied voltage (control voltage to the supplies) for various channels of the instrument. The general shape of each curve is independent of the energy, and MCPs in detection systems are typically operated at the "knee" of this curve. The actual applied voltage used on KIMS is marked on the plot.

detector at different applied voltages. For particle detection, we typically choose an operating voltage that approaches the point of maximum sensitivity without saturating the plates. This saturation can be seen in Fig. 3.8. The plates saturate when very little signal gain is obtained for an increase in voltage. An MCP can be easily damaged if too high of a voltage is applied. (Operating at these voltage levels is safe at the recommended operating pressure of 10^{-6} torr.) From this plot, we are able to determine the optimal operating potential by selecting a voltage just above the "knee". In the case of the MCPs used in the KIMS instrument, the value turns out to be -1.6 kV, or 800 V per plate.

3.2 Calibration

Full calibration of a particle instrument requires measuring the instrument response across a full range of energies using ion beams of known current and energy. KIMS was calibrated using an un-calibrated ion source, limiting knowledge of the absolute densities being measured. The ion source could, however, accurately determine the range of energies measured at each anode pad, leaving the absolute densities somewhat undetermined (though the actual densities will be at least as great as that measured by KIMS). Though not ideal, this time and cost-saving measure is helped by the presence of a second ion instrument on the payload. Cross calibration between the two instruments over their overlapping energy range can provide knowledge of the absolute scale of ion fluxes being measured. In any case, the primary interest in this mission was the relative profile of the energetic ions with height, and the absolute fluxes are not essential.

The nominal energy ranges quoted for KIMS, 100–1000 eV and 1–10 keV, correspond to the energies of protons (H^+) with a momentum per charge that leads to the ion reaching the detector. Because the ion source could only be used with an accelerating voltage of no more than 3 kV, helium (He^+) gas was used for calibrating the instrument. Because the position on the detector scales as $\sqrt{mE}$, this test is equivalent to measuring the response with H^+ of energies up to 12 keV, which is adequate to cover the full range of the KIMS instrument.

The response of both the KIMS-A low energy and KIMS-B high energy sides of the instrument can be considered identical, assuming uniform magnetic fields and

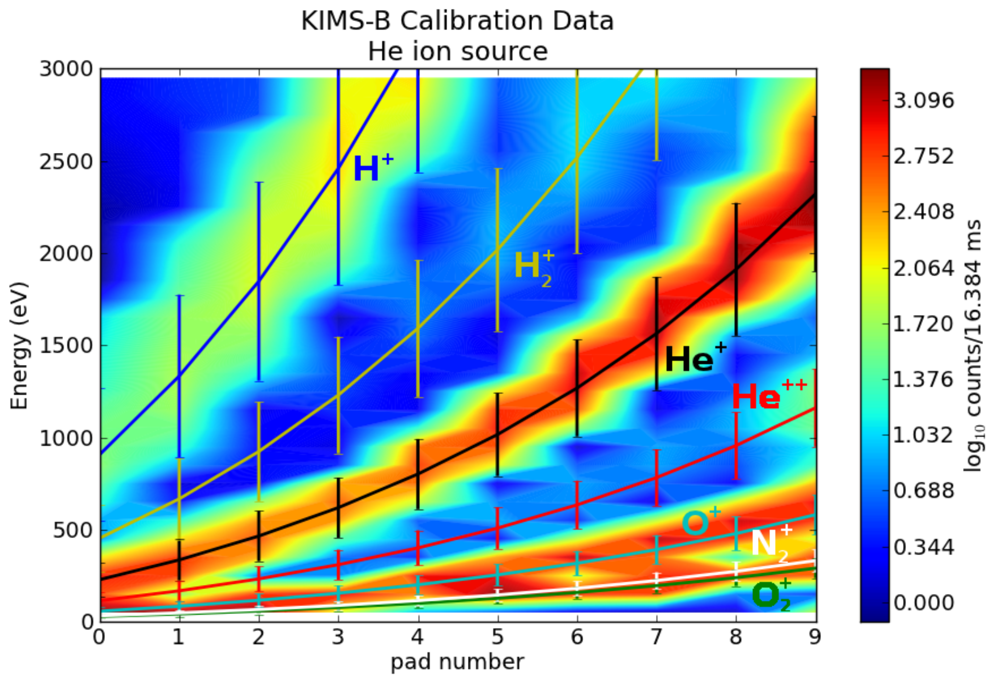


Figure 3.9: Calibration data from the high-energy side of the KIMS instrument. Overlayed are the simulation results for various background gas elements.

identical anode pad configurations. Using He^+ has the disadvantage of lowering the equivalent energy range for KIMS-A to 25–250 eV, a regime where the ion source is less reliable. Fortunately, initial testing with hydrogen gas (H_2^+) between 50 and 500 eV indicated the expected response found in the model, and so this assumption is not unreasonable. For brevity, then, only the results for the KIMS-B calibration are presented here.

The calibration data, shown as a contour plot in Figure 3.9, show excellent agreement with the simulation results. Again, model calculations for various gasses are overlaid on the contour plot. The strongest signal is, by far, the He^+ used as the base gas, though other background gasses do appear in the instrument signals. It should also be noted that the widths of energies measured at each pad are in good agreement with the model.

These data were taken with the ion source beam directed at the aperture with no azimuthal component (ie. at 0°). Data were also taken with the ion source at -5°, +5°, and +10°. These data sets show nearly identical curves with reduced count rates at the extreme angles. As found in the simulation, the azimuthal acceptance range designed into the instrument is small enough to not significantly affect the instrument response, but large enough to provide a sufficient number of counts during flight for good statistics. The solid angle subtended by the view of the KIMS instrument is approximately 0.075 sr.

3.3 Integration Testing

3.3.1 External Magnetic Field

In addition to the internal magnetic fields, the external fields were measured. These measurements were performed at the magnetic calibration facility at NASA Goddard Space Flight Center. Sensitive magnetometers were placed at distances of 20, 40, 60, and 80 cm from the center of the KIMS instrument (see Fig. 3.10). The 3-axis data from the magnetometers were combined to obtain magnetic field magnitudes at these distances. There are noticeable fringe fields at the surface of the instrument case, but for reasonable distances (other payload instruments were not within 30–40 cm of the KIMS instrument) the fields fall as $\frac{1}{r^3}$, characteristic of a dipole field. The weak field at these distances indicates the effectiveness of the

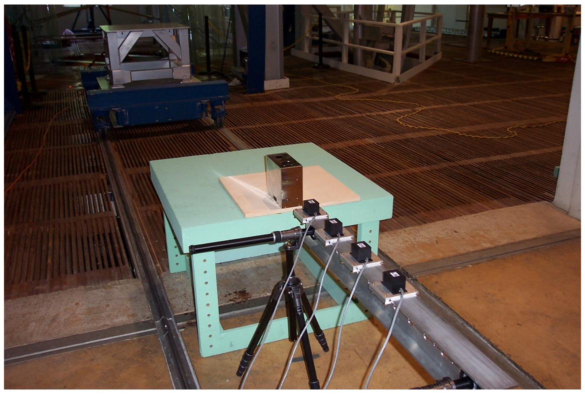


Figure 3.10: Photo showing placement of the magnetometers during the stray field measurement.

shielding from the magnetic circuit and μ-metal case.

The external fields can be fit to a function of the form

$$B = \frac{a}{x^3} + \frac{b}{x^2} + \frac{c}{x} + d \tag{3.1}$$

The measured external fields are well fit with the coefficients $a = -4.02\times10^6$ nT$\cdot$cm^3, $b = 1.79 \times 10^6$ nT $\cdot$ cm^2, $c = -2.06 \times 10^4$ nT $\cdot$ cm, and $d = 96.7$ nT. This fit gives an accurate representation of the near field of the KIMS instrument to a distance of about 100 cm.

A second fit to a roughly dipolar form is given by:

$$B = \frac{a}{x^3} + \frac{b}{x^2} \tag{3.2}$$

where $a = 24.8 \times 10^6$ nT $\cdot$ cm^3 and $b = 0.4 \times 10^6$ nT $\cdot$ cm^2. As seen in Fig. 3.11, this second fit gives a good description of the external fields for distances greater than 40 cm, with an accuracy of better than 10 nT.

While these fits do not show the direction of the magnetic field, they give an idea for overall offset the instrument will cause to the instruments on the payload.

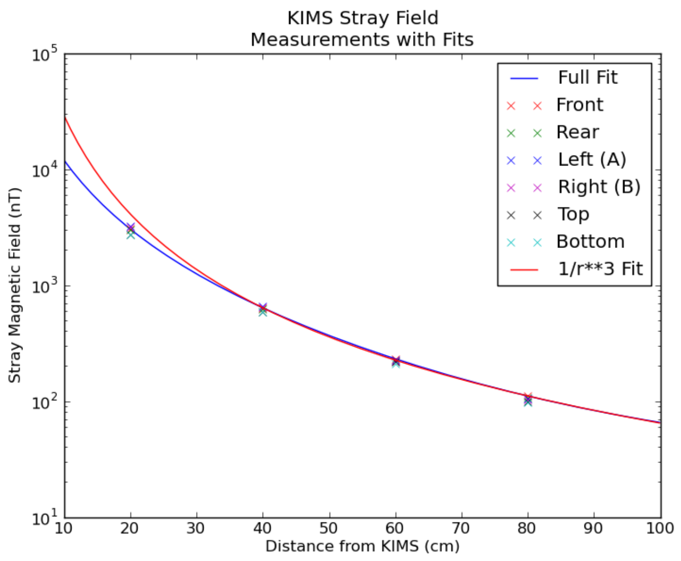


Figure 3.11: Fits to stray field measurement results. These values are absolute magnitudes and do not indicate field direction.

Near the instrument, this offset is roughly comparable to the typical 30 μT of the Earth's magnetic field in the ionosphere. However, since the instrument is designed with permanent magnets, the offset is constant and easily measured at the magnetometer locations so that the offset may be subtracted from their measurements.

3.3.2 Vibration

The KIMS instrument was subjected to vibration tests individually and on the RENU payload. A typical flight profile for a Blackbrant XII rocket was used for the test, with accelerations up to 12 times that of Earth's gravity per Hz. To prevent contamination of or damage to the MCPs, the vibration tests were done with glass slides in place of the MCPs. Since MCPs have shown to be robust in vibration tests of previous instrument designs, this substitution was satisfactory for meeting the overall payload requirements. The slides were cut to match the dimensions of the MCPs, and no issues with vibration were found. Since the orientation of the MCPs is such that the trajectory lies along their long axis, the strongest anticipated vibrations would be along the most rigid dimension of the plates. Because the slides showed no damage or shifting, the KIMS detector stack design proved to be sufficient to keep the microchannel plates intact through the launch of the sounding rocket.

3.3.3 Electronics

During integration testing, the interface between the KIMS instrument and the telemetry module of the sounding rocket was also examined. This test simply ensured that the communication between the KIMS telemetry boards and the payload PCM stack was done properly. With the telemetry board test patterns initiated, the expected signals were reported through the PCM stack, indicating good handshaking and data transfer for the instrument.

3.4 RENU

The KIMS instrument was mounted on the lower bulkhead of the RENU payload with a view oriented radially outward (directed slightly up due to the designed field of view). RENU launched from the Andøya Rocket Range in Andenes, Norway on 12 Dec 2010 with a nominal trajectory over Longyearbyen, Svalbard. The nominal apogee was expected to be 473.2 km altitude. Further details about the launch criteria and conditions can be found in Appendix D.

Unfortunately, a failure in the nose cone ejection prevented RENU from obtaining the data. While the loss of the payload is unfortunate, there are other means of examining the neutral upwelling phenomenon that led to the development of another study using satellite data to examine the importance of various electrodynamic energy sources in neutral upwelling and ion outflow.

Chapter 4

Data Analysis Background

The primary problem in understanding the neutral upwelling phenomenon is a lack of data observing the various electromagnetic energy inputs to the ionosphere in conjunction with neutral upflowing events. While it would be preferable to make such measurements *in situ*, which is what the RENU project hopes to accomplish, there is another means of observing this phenomenon indirectly using statistical correlations of parameters measured by satellites. While no satellite exists that can observe all of the parameters of interest to this particular phenomenon at the necessary orbital configuration, a careful study using multiple spacecraft can provide insight into the most likely places to search for the cause of the neutral upwelling phenomenon.

As the RENU project was unsuccessful at providing meaningful data for the study, we will turn to examining this phenomenon using data acquired by two satellites: the Fast Auroral Snapshot Explorer (FAST) and the Challenging Minisatellite Payload (CHAMP). Both of these missions are highly successful and have already provided significant results to the space science community. (See for example [30], [31], and [32] for examples of FAST results, and [33], [34], and [35] for examples of CHAMP results.)

It is worth noting that this study cannot, by its very nature, substitute for direct (prefereably *in situ*) measurements. A correlation study of neutral upwelling with various energy inputs cannot determine the sources of the phenomenon, as correlation alone does not imply causality. The benefit of such a study, rather, lies in pointing to features in the ionosphere that appear in conjunction with upwelling events. By determining how other parameters of the ionosphere change when the neutral density in the cusp increases, we have information about the particular measurements that may be most beneficial in determining the overall cause.

4.1 Satellite Background

4.1.1 FAST

The FAST satellite was the second mission chosen for NASA's Small Explorer Satellite Program (SMEX). FAST was launched on 21 August 1996 into a highly elliptical orbit with an inclination of 83°. FAST was deployed in a reverse cartwheel configuration[1] with a 5 s period. The instruments included on the spacecraft were a leap forward in the technologies used to study auroral phenomena. The instruments provide a full, 360° view of plasma particles in the spin plane of the spacecraft at a much faster data rate than had been done previously [36].

The FAST electric field instrument consists of six booms. Four booms extend 28 m radially from the spacecraft, each with two spheres located at the end and at 5 m in from the end. The other two booms extend along the spacecraft's spin axis

[1]A reverse cartwheel configuration has the spacecraft spinning opposite the direction of travel. As such, the spin angular momentum vector points opposite the orbital angular momentum vector.

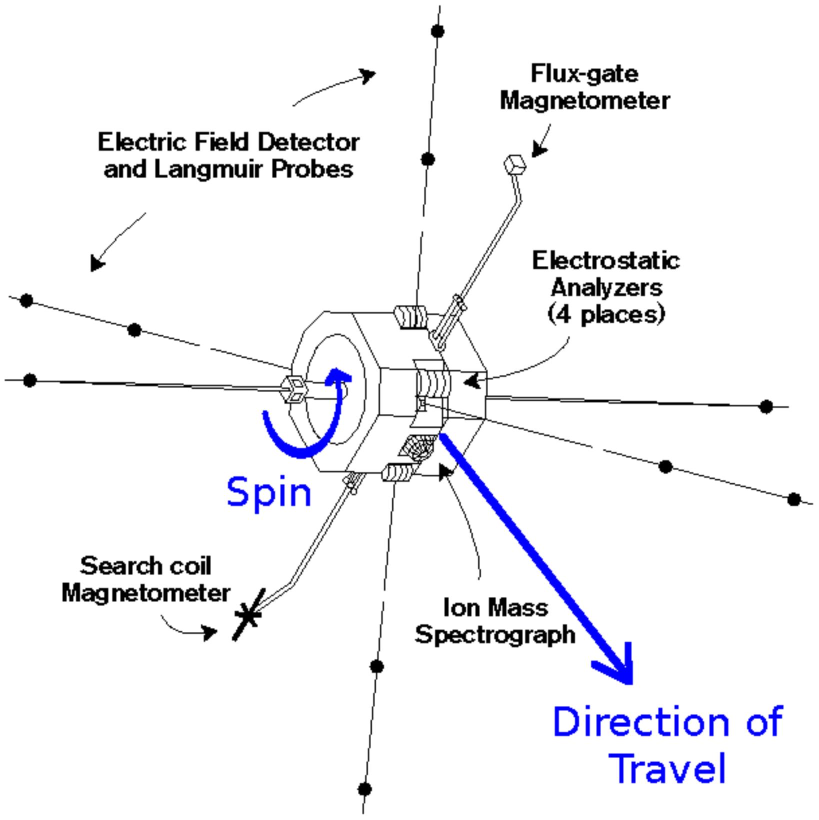


Figure 4.1: Diagram of the FAST satellite and its instruments. (Figure taken from [36].)

3.8 m from the spacecraft center. Each of these two booms have only one sphere. On deployment, one of the four radial booms did not extend fully [37].

The FAST magnetic field instrument uses a three-axis fluxgate magnetometer for measuring DC magnetic fields and a three-axis search coil magnetometer for measuring AC magnetic fields. Both magnetometers are placed on 2 m booms extending radially from the spacecraft [38].

FAST also includes a set of 16 electrostatic analyzers for measuring charged particle energy distributions. Sets of four top-hat style analyzers are distributed at 90° intervals around the spacecraft. Each analyzer has a 180° field of view, providing a full 360° view at all times for the FAST spacecraft with $\sim 90°$ overlap between adjacent instruments. The 180° field of view is separated into 16 individual segments in each spectrometer. Each set of four includes an ion spectrometer, an electron spectrometer, and a pair of electron spectrometers operating in conjunction with the other sets as an electron spectrograph [39]. Further details about the particle instruments are provided in Section 5.2.3.

4.1.2 CHAMP

The CHAMP satellite was a project developed at the GeoForschungsZentrum (GFZ) in Potsdam, Germany, with sponsoring groups including NASA. It was launched 15 July 2000 into a nearly circular polar orbit with a nominal altitude of 454 km. CHAMP was 3-axis stabilized, providing a consistent reference system for the instruments on board. The goal of CHAMP was to provide measurements of the gravitational and magnetic fields of the Earth as well as atmospheric density and composition. The extremely sensitive accelerometer on CHAMP provides precise data on the forces exerted on the satellite that are used to measure both gravity fluctuations and atmospheric density through measurements of satellite drag [40].

The primary instrument of interest for this study on CHAMP is the STAR accelerometer. The instrument design consists of a proof-mass electrostatically suspended in a cage. The instrument applies voltages to keep the mass translationally and rotationally stable. The applied forces are directly counter to the outside forces acting on the satellite. The STAR instrument was placed at the center of mass of the CHAMP satellite to remove the influence of gravity on it. It measures the very small changes in gravitational acceleration and forces caused by air drag, the Earth's albedo, and solar radiation.

The STAR accelerometer is aligned with one axis along the spacecraft direction of travel. The measured acceleration along this axis provides the data necessary for determining the density of the atmosphere by the drag felt in this direction. There are detailed methods of deriving atmospheric density from this data [41], however a simpler model as used by Lühr *et al.* [20] is sufficient for this study. In the simple model, drag in the direction of travel is proportional to density by

$$d = \frac{1}{2}\rho c_f \frac{A}{m} v^2 \tag{4.1}$$

The values c_f (drag coefficient), A (cross sectional area of the satellite), and m (mass of the satellite) are all constant, and so for a constant velocity, d is proportional to ρ, and the decceleration measured in this direction is a sufficient measure of the

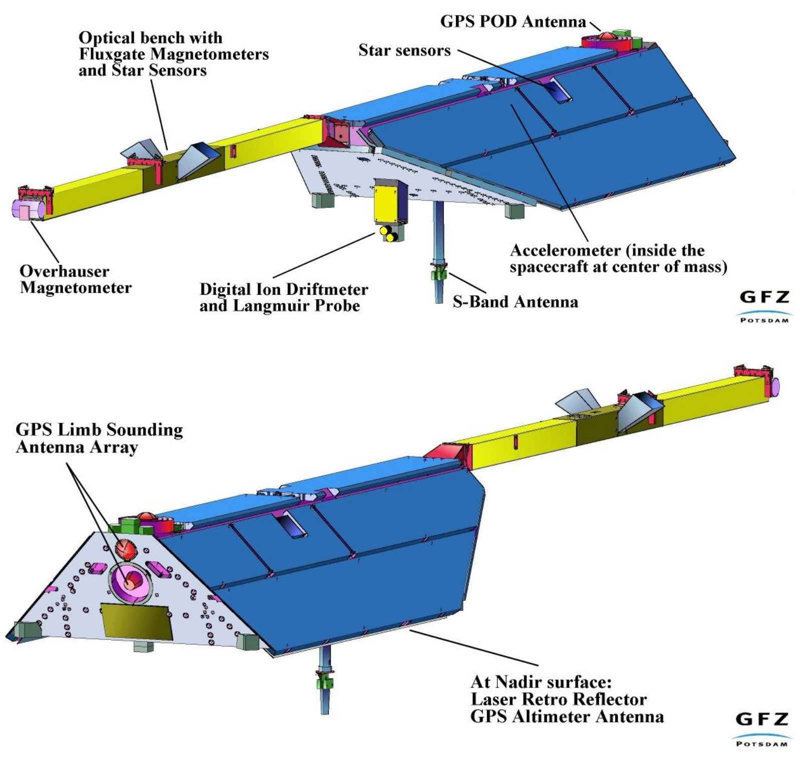

GPS POD Antenna
Optical bench with Fluxgate Magnetometers and Star Sensors
Star sensors
Overhauser Magnetometer
Digital Ion Driftmeter and Langmuir Probe
S-Band Antenna
Accelerometer (inside the spacecraft at center of mass)
GFZ POTSDAM
GPS Limb Sounding Antenna Array
At Nadir surface: Laser Retro Reflector GPS Altimeter Antenna
GFZ POTSDAM

Figure 4.2: Diagram of the CHAMP satellite and its instruments. (Figure credit GFZ Potsdam.)

atmospheric density.

4.2 The Strangeway Study

On 24–25 September 1998, a particularly interesting ion outflow event was observed by the Polar satellite [42]. This event was also observed by the FAST satellite, along with strong field-aligned currents in the ionosphere [43]. Using the electric and magnetic field data provided by the instruments on FAST, it was observed that the outflow exhibited a strong correlation with Poynting flux in the direction of the local magnetic field. An effort was then made by Strangeway [44] to examine the correlations of this particular event with other parameters, particularly in terms of precipitating electron flux.

These two studies used a collection of 33 orbits of the FAST satellite, centered about the September 1998 outflow event. To filter out high frequency field fluctuations, Strangeway averaged the field data over 4 s intervals, then interpolated the averaged data to 1 s intervals. Only data at times where ion outflow was measured in each orbit were included. The selected times were found to correspond to the satellite passing over the dayside cusp. The parameters used in the statistics were also averaged over the entire selected window of each orbit, giving a single data point for each of the 33 orbits in the linear regression.

Because the study was done on such a distinct and limited time period, there are many unanswered questions about these correlations. Do the correlations persist over time, or are they singular to this particular event? With the aggressive averaging used in the study, would the results change with changes in the averaging periods, or inclusion of more data in each average? And, most particularly, what is the change in neutral density during such periods?

The statistical study presented in this dissertation addresses these questions and expands on Strangeway's work to examine these correlations in greater detail. While it is unfortunate that the CHAMP satellite was not in orbit in 1998, preventing the examination of the neutral density during the September 1998 event, there are a number of time periods where the orbits of FAST and CHAMP overlap sufficiently to get meaningful information from of a cross-study between these two data sets. The month of July 2002 proves to be a particularly good period for study, as the orbital planes of the two satellites are within 5° of each other, a number of neutral upwelling and ion outflow events were observed, and data are readily available from both satellites during the month.

Unfortunately, one of the FAST satellite's electric field probes developed a fault in September, 2001. Electric field data are available during the time intervals studied, however the faulty probe compromises their integrity. While the faulty data make a correlation result for Poynting flux values inconclusive during this time period, Strangeway found that Poynting flux and electron precipitation density themselves show a correlation. This suggests that particle data should be sufficient in this analysis.

4.3 Ion Outflow

This study correlates energy inputs with neutral upwelling and ion outflow. Ion outflow is a phenomeon that occurs at the polar cusps when ions are heated to energies such that they are able to escape Earth's gravitational field. This mechanism can be compared to atomspheric, or Jeans, escape [45].

In ion outflow, a stream of charged particles (primarily oxygen) flow along magnetic field lines out of the cusp, but at a slight angle of a few tens of degrees. A satellite passing over the cusp would observe a maximum ion flux at pitch angles on either side of the direction anti-parallel to the local magnetic field, and a low flux directly anti-parallel to the field line. This resulting pattern is characteristic of what is referred to as an ion conic [46], [47]. (See Fig. 4.3 as an example of these signatures.)

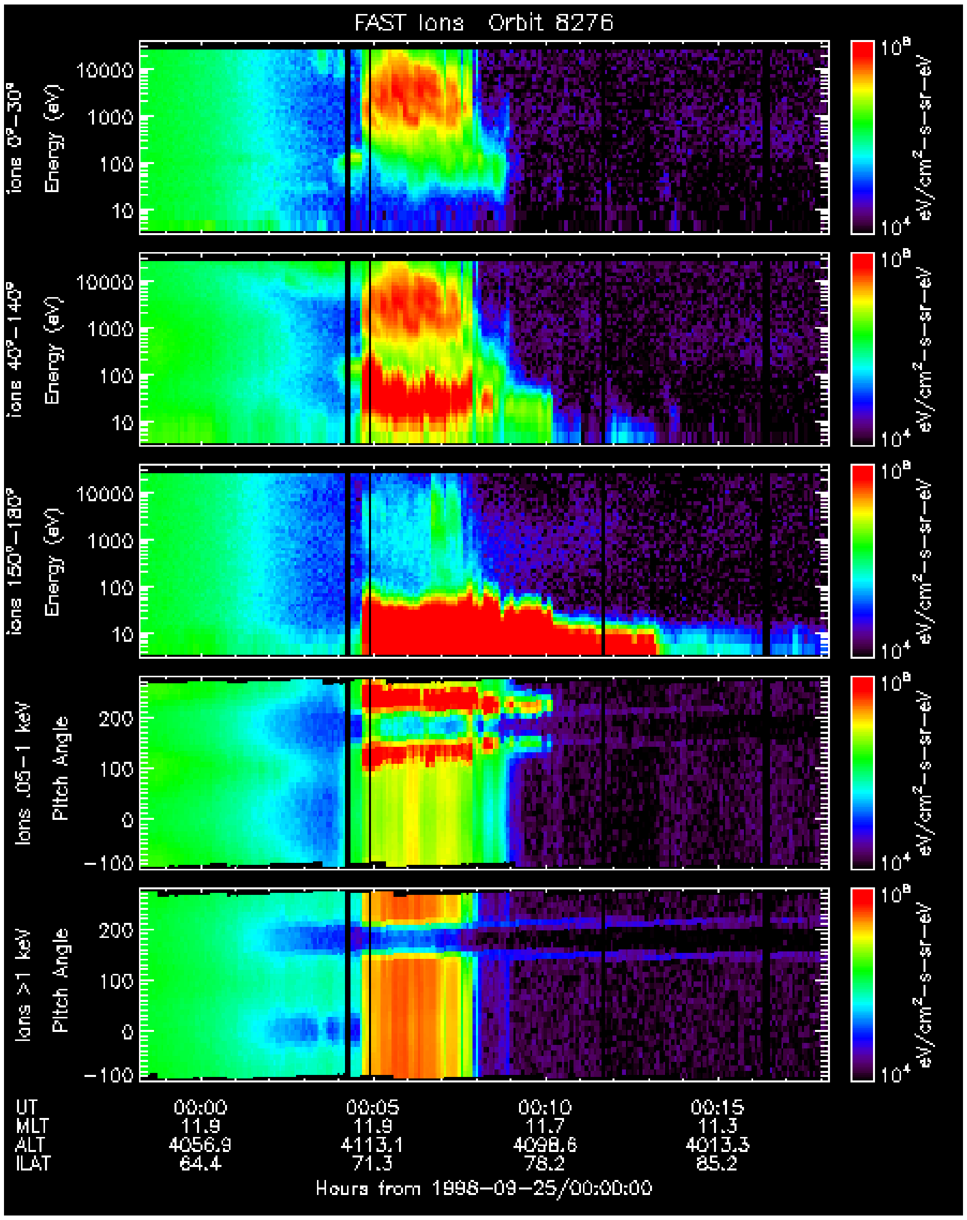


Figure 4.3: An example of a pitch-angle spectrogram of ions in the cusp, from FAST 9/25/1998. The first three panels show energy spectra at directions 0°–30°, 40°–140°, and 150°–180° from the local magnetic field direction respectively. The following panels show the pitch angle spectra for low energy (< 1 keV) and high energy (> 1 keV) particles. The 180° direction is anti-parallel to the local magnetic field, or pointing roughly upward from the Earth's northern pole. In the pitch angle spectra, a reduced count of particles is observed at all energies in this direction. However, in directions a few tens of degrees to either side, a large flux of low energy particles is observed. This pattern is characteristic of an ion conic.

Ion outflows reach high enough altitudes in the ionosphere to be injected into the plasma sheet and ring current of the magnetosphere [48]. Some with sufficient energy even leave the Earth's magnetosphere completely, joining the solar wind as "pick-up ions" [49]. For oxygen atoms at these altitudes, the required energy for escape (assuming the Jean's escape mechanism) is around 10 eV, which corresponds to the typical energies measured in ion outflow streams.

4.4 Ionospheric Energy Sources

The sources of energy that cause ion outflow have been associated with electrodynamic systems, particularly from waves in the aurora [50] and Poynting flux from the electric and magnetic fields in the ionosphere [43]. Outflow has also been associated with electron precipitation in the cusp [51].

The unexpected result of Strangeway *et al.* showed that these three parameters all show strong correlations with the ion outflow flux over the 33 orbit period. The parameters also exhibited strong correlations with each other. These correlations indicate the possibility of a power law scaling between the energy inputs and the ion outflow. As it is statistical in nature, this result does not necessarily mean that the particular inputs cause ion outflow, but rather that there exists some connection between them that can be described mathematically.

4.5 Statistics of Ionospheric Energy Sources

Three electrodynamic systems have shown to be strongly correlated with ion outflow: Poynting flux; electron precipitation; and extremely low frequency (ELF) waves. ELF waves are a known mechanism for heating ionospheric oxygen sufficiently for outflow to occur [52], and heating processes such as Joule heating or electron ionization are known to lead to ELF waves, indirectly linking Poynting flux and electron precipitation with ion outflow [43]. It comes as no surprise, then, that the Poynting flux and electron precipitation would show good correlation with ELF waves.

Surprisingly, both Poynting flux and electron precipitation show good correlation with ion outflow directly, and with each other. Strangeway *et al.* used the field and particle flux data from the FAST satellite to examine the correlation between Poynting flux, electron number flux, and electron energy flux with ion number flux, as seen in figure 4.5. Of these, only electron energy flux exhibited weak correlation. However, by using dimensional analysis, a parameter using the electron energy and number fluxes was established to describe the electron precipitation density within the cusp [44]. This parameter is given by

$$n_{ep} \propto f_{en}^{3/2} / f_{ee}^{1/2} \tag{4.2}$$

where f_{en} is the electron number flux in cm$^{-2}\cdot$ s^{-1}, and f_{ee} is the electron energy flux in mW/m^2. Correlation of this parameter with the ion number flux proves to have a higher correlation than all of the other parameters examined in the Strangeway study (see fig. 4.6). Strangeway found the proportionality constant to be 2.134×10^{-14}. A

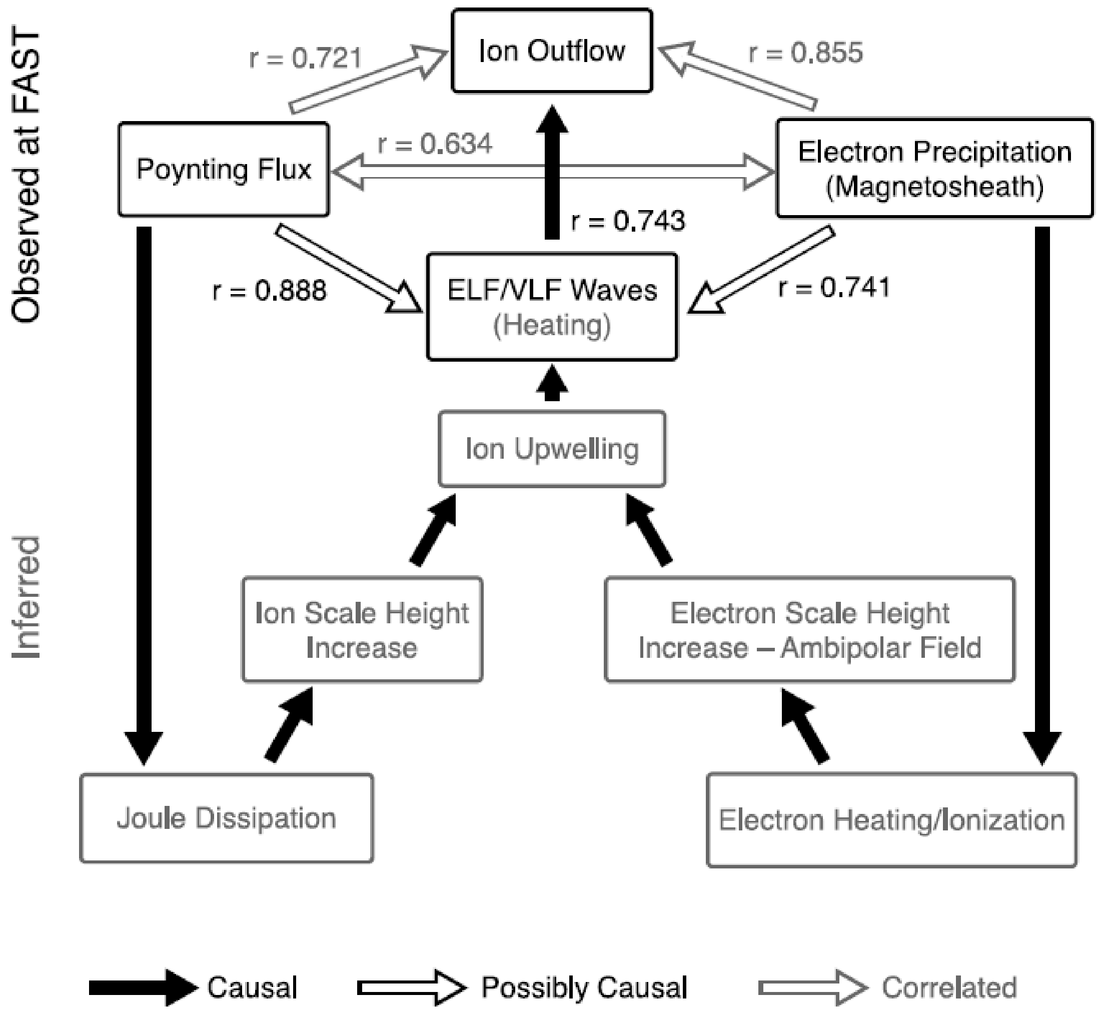


Figure 4.4: Diagram depicting the statistical relationships between energy input parameters in the ionosphere with ion outflow. Solid arrows are used in relationship that are known to be causal, medium-tone outline arrows where a causal relationship might exist, and light-tone outline arrows where a statistical correlation exists between two otherwise independent parameters. The correlation coefficients are reported for correlations with ion outflow measurements, and show good correlations in each. Figure from [44].

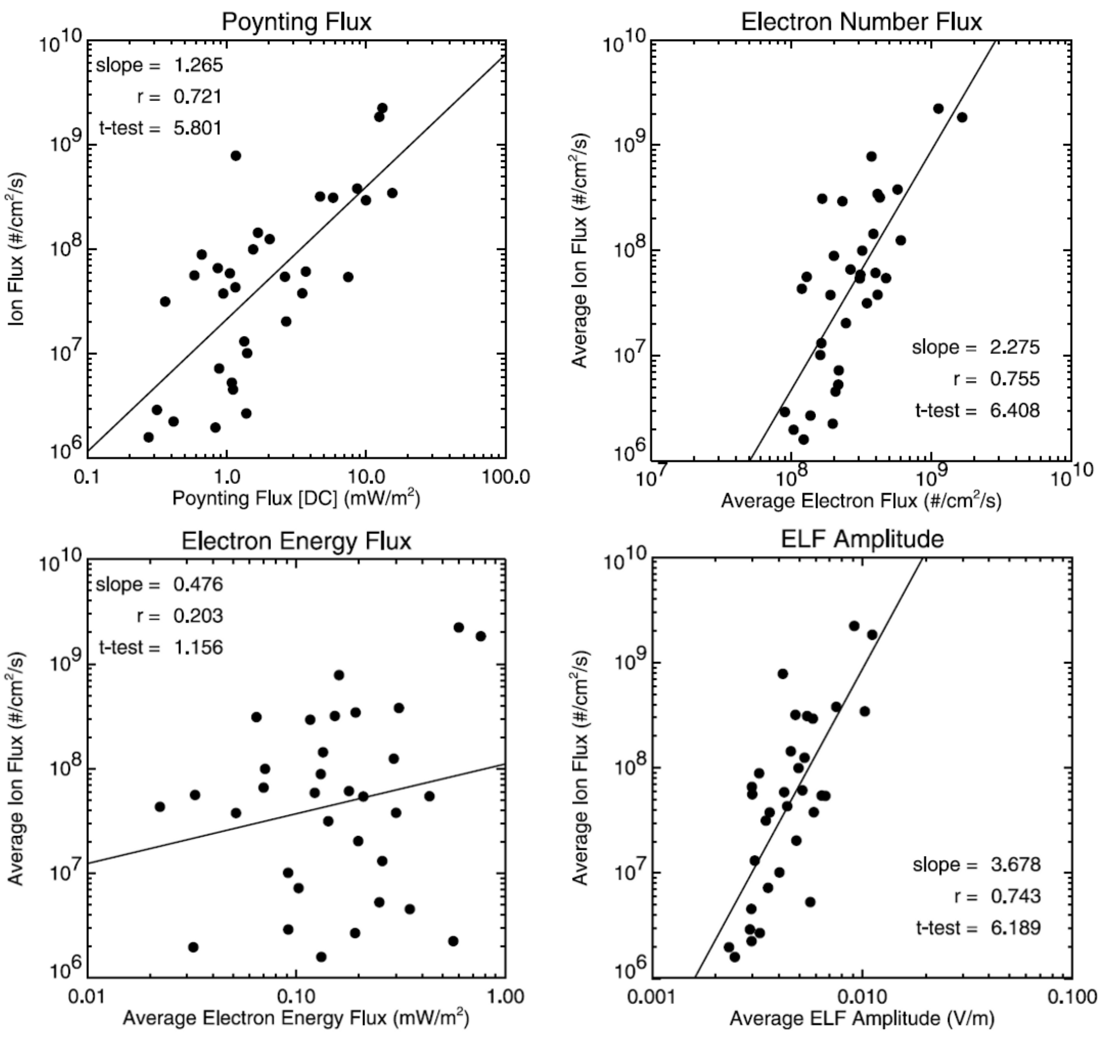


Figure 4.5: Correlation relations of the energy inputs measured by the FAST satellite for 33 orbits centered on the 25 Sep 1998 event (orbits 8260-8292).

similar result was obtained using data from the Polar spacecraft (at a higher altitude then FAST) for the year 2000 [53].

In this portion of the dissertation, the consistency of these correlations are examined over a larger number of orbits and at different periods of time. Because of the known relationship between Poynting flux and electron precipitation with ELF waves, and because the electron precipitation density parameter encompasses both the number and energy fluxes, we limit our study to correlations with the Poynting

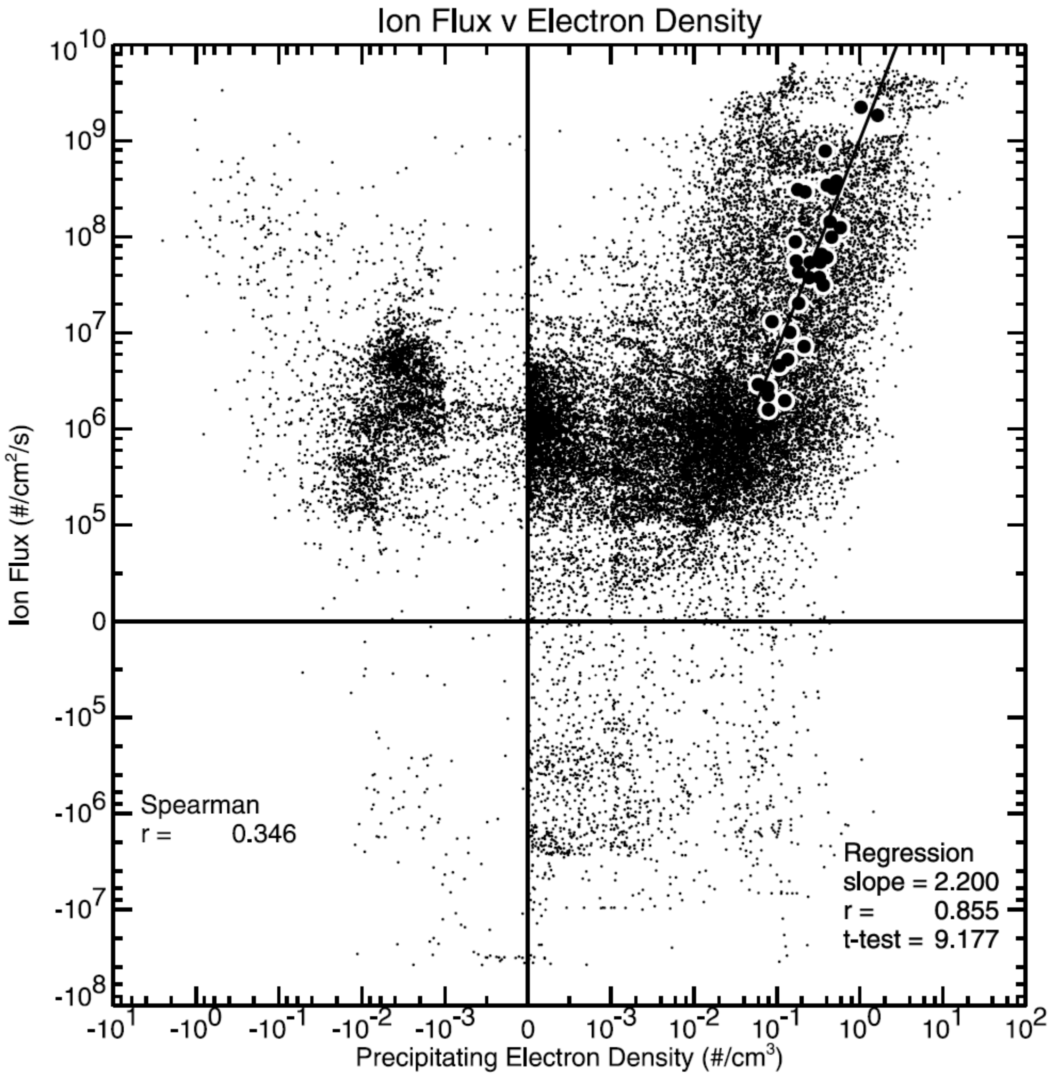


Figure 4.6: Correlation of the electron precipitation density parameter n_{ep}. This plot uses a symetrical logarithm plot, with a linear scale for the intervals closest to zero and a logarithmic scale for the other intervals. Negative values of the ion flux indicate in-flow as opposed to outflow, and negative values of the electron precipitation density indicate upwelling as opposed to precipitation. Individual data points and the orbital averages are both plotted, showing not only the excellent correlation obtained for n_{ep}, but also the consistency of the averages in comparison to the individual data points.

flux and electron precipitation density.

4.6 Statistics in Neutral Upwelling

From the measured values of ion outflow and energy fluxes with the FAST satellite, it is apparent that there is a large discrepancy in the energy balance from the different sources. The amount of energy available in the ionosphere for heating processes is orders of magnitude greater than the energy required to obtain the observed outflows. As the plasma density at these altitudes is significantly less than the neutral density, we expect that some of the remaining energy goes into heating processes of the neutral gas. If this is the case, a correlation between the energy inputs with the neutral density should exist.

Because the FAST satellite does not have an instrument that measures neutral gas density, we use the acceleration data from the CHAMP satellite to obtain this information. It is non-trivial to correlate data sets from two separate satellites. To calculate correlations, it would be ideal to have field, charged particle, and neutral density measurements all measured at the same point in space and instant in time, which is impossibile using two spacecraft. Since the study uses orbital averages, we can instead use averages from similar time periods for each satellite. As long as the orbital inclinations of the two satellites are close together and the phenomena being observed have lifetimes longer than an orbital period, the exact conjunction of time is less essential and this averaging scheme provides an adequate measurement of the process.

In the case of the correlations between FAST and CHAMP, time periods were selected where the orbits of the two satellites lie within 15° of each other. The two time periods chosen are July and February 2002. The July period has the two satellites orbiting within 5°, while the February period is between 5 and 10°. CHAMP, at a 400 km altitude, has a shorter orbital period than FAST. During a single orbit of FAST, CHAMP will pass over the polar cusp at least once but no more than twice. Data from CHAMP is selected by identifying the positions in latitude where FAST data are included for the study. CHAMP data showing a deviation from the orbital trend in the density variation near the same time period as each pass of the FAST satellite over the dayside cusp are selected for inclusion in the analysis. Averages and local maxima of these deviations, or spikes, are compared to the selected orbital averages of the FAST data.

Before being able to use the CHAMP data, the orbital positions of the data need to be compared from within the same coordinate system. FAST coordinates are reported in Solar Magnetic (SM) coordinates, while CHAMP orbits are reported in Geocentric Equatorial Intertial (GEI) coordinates. To work with the two datasets together, the CHAMP orbital data is transformed into SM coordinates. The following section describes the computations needed in order to transform the coordinates, and the code used to do so is found in Appendix E.3.1.

4.7 Geophysical and Geomagnetic Coordinate Systems

In space physics, there are a number of coordinate systems in common use, each used to describe positions, velocities, and directions in terms of some convenient inertial reference frame. For examining the motion of celestial bodies in relation to the Earth, we often use a coordinate system centered on the Earth with an axis fixed to point to the First Point of Aries (or the point in the constellation Aries that is perpendicular to the line between the sun and the Earth on the Vernal Equinox), which we call the Geocentric Equatorial Inertial (GEI) system. In describing events dealing with interactions between the sun and the Earth, however, it is often more convenient to work in a geocentric system with an axis fixed along the line between the sun and the Earth, called Geocentric Solar Ecliptic (GSE). More commonly, most people are familiar with the lines of latitude and longitude used in describing location on the surface of the earth, called the Geographic (GEO) coordinate system. The magnetospheric system is best described in a different set of coordinates, similar to GEI but rotated to the location of the magnetic poles. Two types of coordinates often used in magnetospheric studies are Geocentric Solar Magnetospheric (GSM) coordinates and Solar Magnetic (SM) coordinates.

Each of these systems as well as others are commonly used, and transformations between systems are well understood. Russell has provided an excellent summary describing the systems and the transformations between them [54]. Here we will describe the coordinate systems and transformations pertinent to this particular study.

4.7.1 Geographic Coordinates (GEO)

Geographic Coordinates describe a system centered on the earth with the z-axis through the North geopgraphic pole and the x-axis through the equator and prime meridian. Usually expressed in terms of latitude, longitude, and altitude, these coordinates are commonly used by anyone familiar with geographic maps. This system is often used in transformations between other systems as well.

4.7.2 Geocentric Equatorial Inertial Coordinates (GEI)

For systems that need to reference the earth without its rotation, we switch to a coordinate system where the x-axis does not rotate with the earth. Convention sets the x-axis to point perpendicular to the Earth-sun line on the Vernal Equinox. This direction happens to point toward the constellation Aries, and is known as the First Point of Aries.

Transformation from GEO to GEI is simply a rotation about the z-axis, but the angle of rotation must be calculated based upon the date and time. This angle, known as the Greenwich Hour Angle (GHA) or Greenwich Sidereal Time (GST), can be found with a number of different formulas of varying accuracy. A commonly used formula, given by Russell, is as follows:

$$\theta_{GST} = 279.690983 + 0.9856473354 \times D_J + 360 \times f_D + 180 \tag{4.3}$$

In this formula, D_J is given by the Julian Date, while f_D is the time of day expressed as a fraction of a day. The result is the Greenwich Hour Angle in degrees. The resulting value can be restricted to be within the range of 0—360° by calculating the result modulo 360.

4.7.3 Geocentric Solar Magnetospheric Coordinates (GSM)

For systems related to the Earth's magnetic field, it is useful to have a coordinate system relative to the magnetic poles rather than the geographic poles. In addition, the position of the sun is a convenient fixed point for these systems. GSM aligns the z-axis to be in the same plane as the Earth's magnetic dipole moment. The x-axis points toward the sun.

Transformation from GEI to GSM is linear, however it also requires a calculation of not just the GHA, but also the position of the sun. Again, formulae are available to make this transformation easy, particularly those given by Russell which provide an accuracy to within 0.006° [54].

4.7.4 International Geomagnetic Reference Field

In addition to solar position, GSM requires knowledge of Earth's magnetic field. Unfortunately, the magnetic field is not constant, particularly over long time periods. The magnetic poles are not aligned and drift independently. Models have been developed, however, that allow us to predict the Earth's magnetic field to good accuracy. While it is not a perfect dipole, the dipole term of these models is generally sufficient to locate the position of the north magnetic pole and make the transformation from GEI to GSM.

One of the more commonly used models of the magnetic field (and that used in this study) is the International Geomagnetic Reference Field (IGRF) [55]. As more data are acquired, updates to the model are released periodically. The most recent model is IGRF-11, released in 2010. Strangeway uses IGRF-7 (1995) in his calculations, and high resolution data for the FAST satellite are processed using the same model regardless of the time period selected. To remain in a consistent coordinate system with the orbital parameters provided for the FAST satellite, IGRF-7 is used in transforming CHAMP orbital data for this study.[2]

4.7.5 Solar Magnetic Coordinates (SM)

GSM is particularly useful for studying the magnetosphere and solar wind. For systems closer to earth and more strongly associated with the local magnetic field, a slight adjustment can improve the accuracy. The Solar Magnetic (SM) coordinate system rotates the GSM system about its y-axis to bring the z-axis parallel to the dipole moment. We can then express this system in terms of latitude and longitude, just as we do in Geographic coordinates. In this system, latitude, measured in degrees, is called Invariant Latitude (ILT). Longitude, measured in hours with 12:00 being the line of longitude aligned with the x-axis, is called Magnetic Local Time (MLT). (Note that magnetic noon in this system, the x-axis, is aligned with the sub-solar point, or the point where solar radiation directly impinges the Earth. The x-axis does not, however, point directly at the sun in this system.)

SM coordinates are used in this study, and have the same assumptions and

[2]Further information on and the matrix coefficients used in the IGRF model can be found at http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html (retrieved 27 Sep 2011).

requirements described above for GSM coordinates. A simple way to view the transformation is to take GEI coordinates, and rotate about the z and y axes by the angles given by the longitude and latitude of the north magnetic pole respectively, in that order. Then a rotation about the new z-axis brings the x-axis to point at the sun. Russell gives this transformation as a single rotation matrix as follows:

$$\begin{bmatrix} X_1 & X_2 & X_3 \\ Y_1 & Y_2 & Y_3 \\ D_1 & D_2 & D_3 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}_{GEI} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}_{SM} \tag{4.4}$$

The vectors **X** and **Y** are obtained from the vectors **S** and **D**, which describe the Earth-Sun direction and the dipole direction in GEI, respectively. After calculating the vectors **S** and **D**,

$$\vec{Y} = \frac{\vec{D} \times \vec{S}}{|\vec{D} \times \vec{S}|} \tag{4.5}$$

$$\vec{X} = \vec{Y} \times \vec{D} \tag{4.6}$$

4.8 Summary

The goal of this portion of the dissertation is to perform a statistical analysis relating ionospheric energy inputs to ion outflow and neutral upwelling by using data from the FAST and CHAMP satellites. The techniques are based on Strangeway's work and used to expand the results reported by him. To correlate neutral density data taken from the CHAMP satellite's accelerometer, time periods within the orbital period of FAST are selected for computing the average densities corresponding to the averages from each FAST orbit. The study is limited to two energy inputs: electromagnetic energy (Poynting flux) and electron precipitation density. The results of this study will demonstrate that the power law relationship observed for these inputs with ion outflow is indeed a consistent result and that a correlation exists between ion outflow and neutral upwelling, suggesting a relationship with these energy inputs.

Chapter 5

Data and Analysis Software

In this chapter, we give a description of the analysis techniques and software employed in analyzing the FAST and CHAMP data sets. FAST data are available for download from a website established at the University of California, Berkeley [57]. Data are downloaded by orbit number in the Common Data Format (CDF) [58]. CHAMP data are available at the Information System and Data Center at the Helmholtz Center, Potsdam, Germany [59].

One of the underlying goals of this study was to create a software framework that simplifies the analysis of data from both FAST and CHAMP in examining these correlations. Our hope was to make the process as autonomous as possible and to be able to provide processed data in an easily accessible format. The Python language was chosen to do the analysis of the data, partly because it provides a binary data storage method that makes data products compact, portable, and easily documented. An overview of basic Python tools needed in this study is given in Section 5.1.

The data processing consists of preparing the various spacecraft measurements and interplating each to a common time base. DC electric and magnetic field data are smoothed and interpolated to provide the calculation of Poynting flux, as described in Section 5.2.2. AC electric field data are integrated to examine extremely low frequency waves (see the same Section). Particle data are integrated to provide ion and electron flux measurements using methods described in Section 5.2.3. Particle data used in the analysis are selected based on ion outflow signatures and adjusted when necessary to account for the spacecraft motion. Neutral gas density is derived from acceleration measurements, using methods described in Section 5.2.6. Because these data come from the CHAMP satellite rather than the FAST satellite, special handling is done to identify data for analysis by examining CHAMP orbits over the northern cusp at times corresponding to FAST orbits over the cusp within an orbit period. The processed data are then averaged and analyzed with statistical methods described in Sections 5.2.4 and 5.2.7.

5.1 Python

To meet the goals of the software development, the Python programming language is an ideal environment for establishing a consistent framework. Because of its scripting capabilities, Python is very good at handling the ASCII text format of the CHAMP satellite, something that typical computation enviornments such as Matlab or IDL do not handle well. The recent public release of the SpacePy library made it possible to also work with CDF archives within Python, which was the primary motivation to use Matlab or IDL. The C language also does well with parsing text files and handling CDF, but visualizing and plotting the data would require a great deal of effort. By combining the power of Python with the NumPy [60], SciPy [61], Matplotlib [62], and SpacePy [63] libraries, all of the needed tools to easily handle the data sets are available within a single environment.

Finally, Python provides a binary data storage format that makes it simple to store computational results for further processing later. This format, the “pickle”, allows transferring data and metadata easily from one user to another. The code used in this dissertation (found in Appendix E) was built to be a general framework for using FAST and CHAMP data sets. There are some differences between how the two satellites are handled in the code, partly due to the different data formats and availablity, and partly to explore different ideas of interaction in the inital processing.

Currently, two somewhat incompatible versions of Python are available. This code was developed using the Python 2.7 branch. Ideally, the data products of the code would be useable in either the 2.x or 3.x branches, and indeed the pickle format was designed to be able to do that. However, the NumPy array structure encodes in pickles slightly differently between the 2.x and 3.x systems (as of this writing, at least), and so provision must be made to use the same branch of Python as that in which the data products are made.

5.1.1 Python Structures

The code used in this study make extensive use of two structures provided by the Python framework: the list and the dictionary. A list is simply an ordered collection of data, and is denoted using the bracket:

```
list = [1, 2, 3]
```

The data included in a list can be of any type, and there is no stipulation that each entry in the list must be of the same type. Elements are referenced by index, with the leftmost starting at 0 (ie. `list[0] = 1` and `list[2] = 3` in the above example). Elements can also be referenced using a negative index, with the rightmost starting at -1 (ie. `list[-1] = 3` and `list[-3] = 0` in this example). Portions of lists can be referenced using the slice, with the subset including the starting index and not including the ending index (ie. `list[0:2] = [1, 2]`. The notation `list[:2]` is equivalent to a slice starting at the first (0th) element and including up to index 2. `list[1:]` would start at index 1 and include all the following elements, and `list[:]` includes all elements.

The dictionary is a structure that is similar to the list, but with a user defined index. Dictionaries are denoted with braces. For example,

```
dict = {0:1, 1:2, 2:3}
```

creates a structure similar to the above list, with `dict[0] = 1` and so forth. Users are not limited to integer indicies, however, and the following is equally valid:

```
dict = {'a':1, 'b':2, 'c':3}
```

This example would allow us to reference each element as `dict['a'] = 1` and so forth. The dictionary structure is particularly useful in creating data structures for storage, as saved arrays can be referenced by their function rather than having to specify the arrangement of data in the file.

5.1.2 NumPy

Numerical Python, or NumPy, is an extension to the Python language that provides array-like structures and extensive computational libraries. NumPy arrays can be made of any dimension or size within the bounds of the computer's memory. The array is essential in scientific computing not only for vector calculations, but also for performing operations on any time or spatial series. For example, to scale a standard Python list such as $3 \times [1, 2, 3]$, a loop is required to create a new list:

```
list = [1, 2, 3]
newlist = []
for i in list:
    newlist.append(3*i)
```

NumPy simplifies this process by allowing operations to extend across the array:

```
import numpy
list = numpy.array([1, 2, 3])
newlist = 3*list
```

NumPy also provides essential libraries, such as fft, histogram generation, and various mathematical operations for array computations. The NumPy package can be included in a Python program by importing the entire library or individual parts. `import numpy` will bring in the entire library, accessible by referencing various functions as `numpy.`*`function`*. The notation `from numpy import` *`function`* provides access to a specific function by name directly into the local namespace, rather than having to reference the `numpy` namespace. The code listings in Appendix E make use of the latter method. Imports are done at the beginning of each listing for quick reference to what libraries and functions are needed in the code.

5.1.3 SciPy

Scientific Python, or SciPy, builds on the NumPy package to provide further functions useful in scientific computing. SciPy includes functions for integration, optimization, linear algebra, interpolation, and statistics, to name only a few. Further information on the NumPy and SciPy packages is available, along with binary installation files and source code, at the SciPy website [61].

5.1.4 Matplotlib

Matplotlib is a package that provides 2D plotting capability. Some 3D plot ability is available as well, and will continue to improve as the package is further developed. The syntax and functionality of Matplotlib is similar to that available in Matlab. All graphs and plots provided in this dissertation are created using Matplotlib.

5.1.5 SpacePy

SpacePy is a new package, released publicly under an open source license during the course of this study. The purpose of SpacePy is to provide a Python interface to computational libraries developed for space physics modeling and analysis. For this study, the particular functionality needed is an interface to NASA's Common Data Format. CDF is a binary file format that is used for storing data and metadata in the space physics community. The FAST data is provided as a CDF file, and this package made it possible to work directly with those files rather than having to extract the data and convert it to a different format.

5.1.6 The Python Pickle

The pickle module in Python is a method of converting data objects in memory into a byte stream that can be saved to a disk for later retrieval. Data products in this study are saved as a pickle format which are then used for doing the statistical analysis.

The Python langauge also provides a data structure called a shelf. The shelf is a way of storing multiple pickles in a single file, each referenced in a similar manner to a dictionary. This function is similar to pickling a dictionary structure as is done in this work. Future work would benefit from using the built in pickle shelf for simplicity.

5.2 Methods

5.2.1 FAST Data Processing

A variety of data types from the FAST satellite are made available at the FAST website. The data types used in this study include the despun DC electric field, the despun DC magnetic field, the electron spectrometer survey, the ion spectrometer survey, the low frequency V5-V8 AC electric field, and the orbital parameters. The magnetic field data is built off a model including the IGRF-7, and requires a full orbit's worth of data to ensure its accuracy. Hence, each data type is downloaded by orbit number rather than by a specified time period. Each file is stored in directories

`dcefs_ooooo.cdf`	DC Electric Field
`dcmag_ooooo.cdf`	DC Magnetic Field
`ees_ooooo.cdf`	Electron Survey
`ies_ooooo.cdf`	Ion Survey
`dsp_v58_ooooo.cdf`	AC Electric Field
`orbit_ooooo.cdf`	Orbit Parameters

Table 5.1: Filename conventions used for FAST CDF data files. *ooooo* is given by the orbit number with padded zeros to give 5 characters.

named by orbit number using the filenames given in Table 5.1.

The data types are all measured independently and come with their own time bases. Each must be extracted individually and interpolated to a common time base. The routines in `FAST.py` (Appendix E.1.1) were written to take a collection of orbit numbers as inputs and automatically extract and interpolate all of the data types for that orbit. The following sections will describe the various components of this code.

5.2.2 FAST Field Data

5.2.2.1 Field Processing Tools

Three functions are provided for preparing electric and magnetic field data in this study. These functions are:

`FAST.chauvenet(data, window, weight)`

DC electric field data from FAST often has periodic spikes corresponding to the rotation of the spacecraft. These spikes can be large enough to throw off averages and interpolations, and so the Chauvenet criterion is used to filter these spikes out of the data used in averaging and interpolating. This criterion removes any data points that are farther from the sample mean than a given threshold, usually given in multiples of the standard deviation.

In this function, a window and weight are passed along with the data to be filtered. The window specifies how many points to include in calculating the sample mean, and the weight sets the threshold in standard deviations. The first window/2 points are compared to a set of the first window points, the last window/2 points to a set of the last window points, and all others to a set of size window centered about the point under consideration. Provision must be made to accommodate data with a value of `nan`[1], lest the average and standard deviation are returned as `nan` values. The `nantools.py` routines (Appendix E.4.1) provide this ability. Any data point farther than the number of standard deviations from the sample mean given by `weight` is itself set to the value `nan`. This study used a window of 100 points and a weight of 2 standard deviations when applying this filter. All DC electric field data was filtered.

[1]The `nan` value indicates a non-numerical result, or “not a number”.

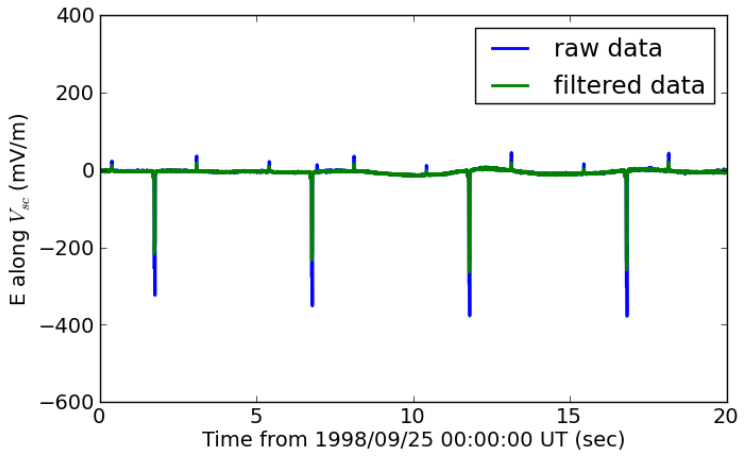


Figure 5.1: DC electric field data from a portion of FAST orbit 8276 showing the effect of applying Chauvenet's criterion with a window of 100 points and a weight of 2 standard deviations.

`FAST.smooth(data, r_data, interval)`

All field data are smoothed, or averaged, to 4 s intervals. The `smooth` function takes the data, the corresponding time series, and the new, smoothed time series to create an averaged array of the data. The new time series is created by making an array with the same endpoints as the raw data, but at intervals of 4 s. A fixed-width window is used in this scheme, averaging only data points between ±2 s. For end data points where data do not exist in the entire window, only data within the window are used.

Smoothing is applied to all the field data types before interpolating to 1 s intervals. The effect of the aggressive smoothing algorithm is to limit data to slowly varying changes in the DC field data.

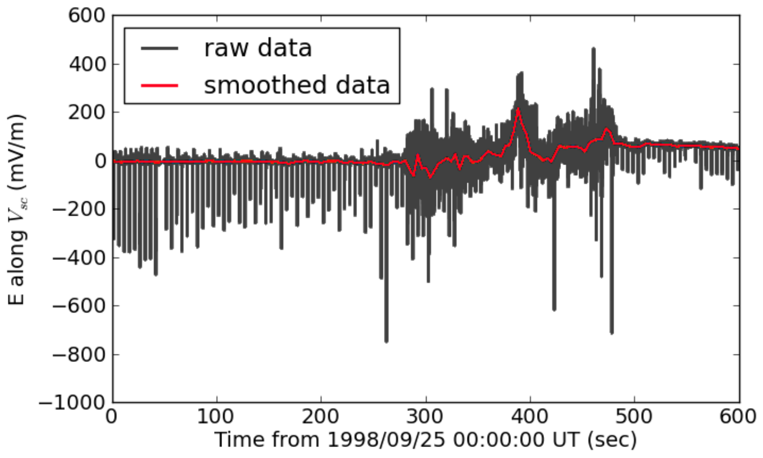


Figure 5.2: DC electric field data from a portion of FAST orbit 8276 showing the effect of applying the the despiking and smoothing algorithms to the raw data. The data are smoothed to 4 s intervals.

`FAST.interpolate(x, y, x_new, width, int_kind)`

An interpolation routine is provided that makes use of the `interp1d` function within `scipy.interpolate`. The `FAST.interpolate` function accepts the raw time and data values (`x` and `y`) and the time base to which the data should be interpolated (`x_new`). In addition, a width is given to limit the amount of data used in the interpolation as well as an optional parameter to specify the interpolation type. (Cubic spline interpolation is used by default.) The width limit was necessary to avoid calculating cubic splines over the entire data set for a single point. A width of 30 s was used in the study, or in other words, 7 data points from the smoothed data were used for each interpolation point.

The time base used for interpolation is obtained from the orbital parameters data of the orbit. These data are given in a 1 s interval time base, and for con-

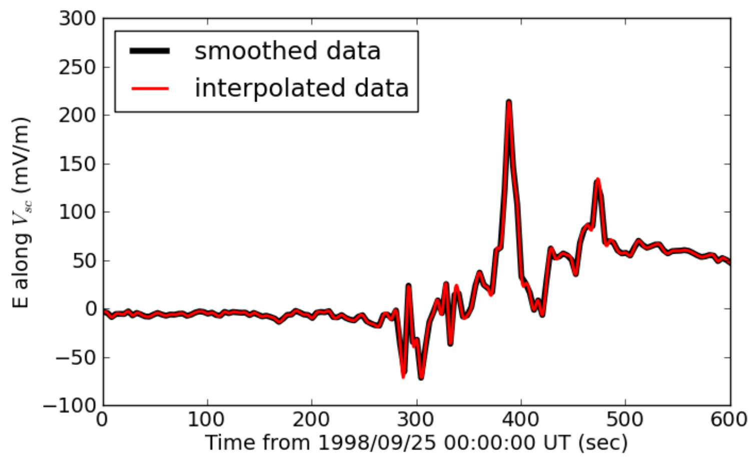


Figure 5.3: DC electric field data from a portion of FAST orbit 8276 showing the effect of applying the interpolation algorithm over 1 s intervals.

venience the same base was used for all other data types. Note that this function will only work if the end points of the raw data are at least width/2 further than the end points of the time base to which the interpolation is done. (Extrapolation is not provided in this code.) To accommodate the time values in the orbit where data are not provided, the function sets the interpolated values to `nan`.

5.2.2.2 DC Electric Field

`FAST.get_E(orbit, t_base, margin, subset)`

The `FAST.get_E` function imports the data from the CDF file corresponding to the specified orbit. Electric field data are sampled at 3.125 ms intervals on FAST. `t_base` is passed to the function from outside, and as mentioned previously the time series from the orbit data was used. A default margin of 30 s was given for the interpolation window. By default the function parses the entire data set, but provision is made to process only a subset of the data if desired. This functionality was not used beyond testing purposes in this study.

From the `dcefs` data, two time limits are found. The `t_min/t_max` pair limit the time values in the electric field data to the end points of the interpolation (extended by margin/2 on either side to accommodate the interpolation). The `ti/tf` pair limit the time values in the interpolation time base where data will be provided. In the context of processing the entire orbit, only the latter pair is needed.

Once the time endpoints are found, the raw data are converted to numpy arrays. The electric field data are filtered with `FAST.chauvenet`, smoothed with `FAST.smooth`, and finally interpolated with `FAST.interpolate`. The interpolated E field at the time values given in `t_base` is then returned. An example of the result is given in Figure 5.3.

A fault in the V5 electric field probe in Sep. 2001 has affected the quality of the electric field data obtained by FAST. The fault causes the data to show oscillations on scales larger than the ambient electric fields. Because of this fault, electric field data from after Sep. 2001 should be considered as compromised.

5.2.2.3 ELF AC Electric Field

`FAST.get_ELF(orbit, t_base, margin, subset)`

AC electric field data is processed by the `FAST.get_ELF` function. The function parameters being passed indicate an intentional similarity in how this function is

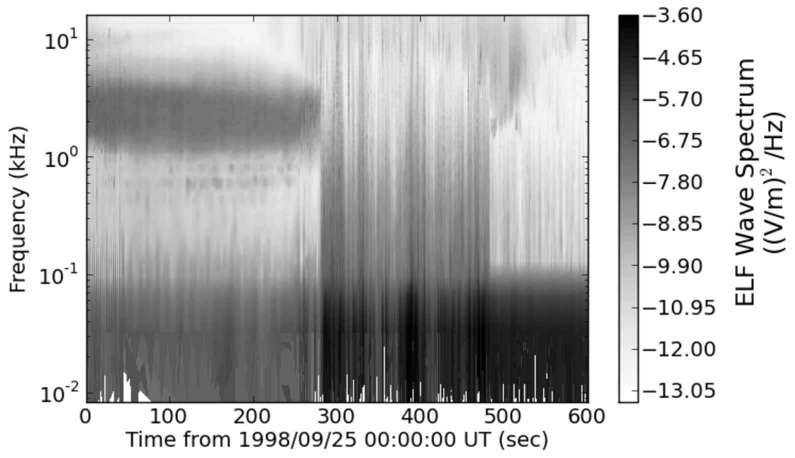


Figure 5.4: Spectrum of the AC electric field from FAST orbit 8276.

written to the `FAST.get_E` function, and so those details will not be rediscussed here.

The `dsp_v58` data provides a power spectrum at frequencies between 0 and 16 kHz (see Fig. 5.4). The power spectrum is integrated over frequency using the trapezoid method. A separate smoothing algorithm is used to smooth the ELF data by replacing each data point with the average of the five points centered about it. (The separate routine was done to avoid a complication in the differences of how the data type is provided.) Finally, the data are also interpolated to the orbit time base. An example of the result of this function is given in Figure 5.5.

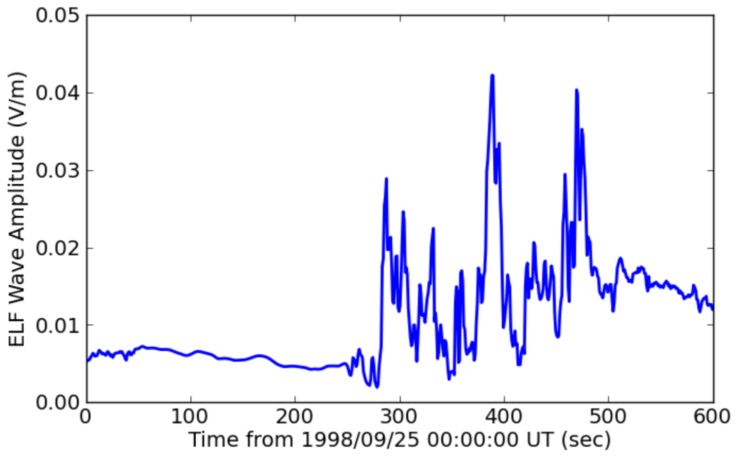


Figure 5.5: Interpolated data from the DSP V5-V8 power spectrum from a portion of FAST orbit 8276.

5.2.2.4 DC Magnetic Field

`FAST.get_B(orbit, t_base, margin, subset)`

The FAST magnetometers provide three-axis magnetic field data as deviations from Earth's magnetic field as provided by the IGRF model. DC magnetic field data are processed by the `FAST.get_B` function. This function is analogous to the `FAST.get_E` function, and uses a nearly identical routine. For the magnetic field, however, Chauvenet's criterion is not needed, and three components of $\vec{B}$ are provided. $\vec{B}$ field data are sampled at 7.8125 ms intervals. The data are smoothed to 4 s intervals to remove high frequency changes and interpolated to 1 s intervals just as the $\vec{E}$ field data are.

The FAST magnetic field data provided are the measured deviations from the model provided by IGRF-7. The coordinate system used is a spacecraft-centric, field-

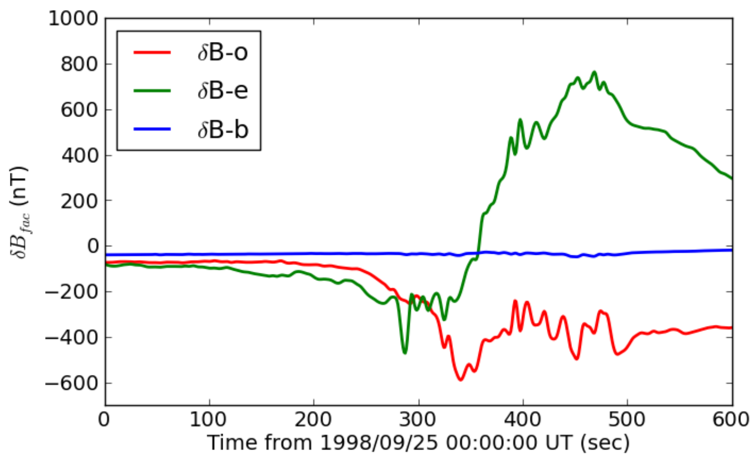


Figure 5.6: Interpolated data of the magnetic field deviation from a portion of FAST orbit 8276. The three components given are along the local magnetic field ($\hat{b}$), perpendicular to the magnetic field and the spacecraft-Earth line pointing eastward ($\hat{e}$), and the right-handed complement pointing outward, or away from the Earth ($\hat{o}$).

aligned, right-handed coordinate system using the notation $\hat{o}, \hat{b}, \hat{e}$. The $\hat{b}$ component points in the direction of the local magnetic field (downward, in the case of the northern cusp). The $\hat{e}$ component points perpendicular to the plane defined by the magnetic field and the radius vector to the spacecraft, or perpendicular to the local magnetic meridian with the positive direction pointing eastward. The $\hat{o}$ component completes the triad, and points "outward", meaning in a northerly direction in the northern hemisphere and a southerly direction in the southern hemisphere. Note that for a given magnetic field line, a transition over the cusp results in a rotation of this coordinate system by 180° about the $\hat{b}$-axis.

An example of the result from `FAST.get_B` is given in Figure 5.6.

5.2.2.5 Poynting Flux

`FAST.get_EBS(orbit, t_base, margin)`

The Poynting flux is found by

$$\vec{S} = \frac{1}{\mu_0}\vec{E} \times \vec{B} \tag{5.1}$$

The FAST spacecraft only measures the electric field along the field direction and along the spacecraft trajectory. At the cusp, the spacecraft velocity is nearly perpendicular to the magnetic field, and so the Poynting flux along the magnetic field lines can be found by

$$S_z = \frac{1}{\mu_0} E_{V_{sc}} \cdot \delta B_e \tag{5.2}$$

The `FAST.get_EBS` function gets the interpolated fields from `FAST.get_E` and `FAST.get_B`, and calculates S_z according to formula 5.2. The electric field along the trajectory, the three components of the magnetic field deviation, and the Poynting flux are all returned as interpolations along a common time base.

5.2.3 FAST Particle Data

FAST particle data are provided as flux per energy · solid-angle at various energies over the 64 detector segments covering the full 360° view with a 6° resolution around the spacecraft spin axis. The data for each species are aggregated from the four ion and electron spectrometers around the spacecraft. Energies are measured in 48 steps ranging from 3 eV to 25 keV for electrons, and 4 eV to 30 keV for ions. The spacing between samples was chosen for an approximately constant ΔE/E for

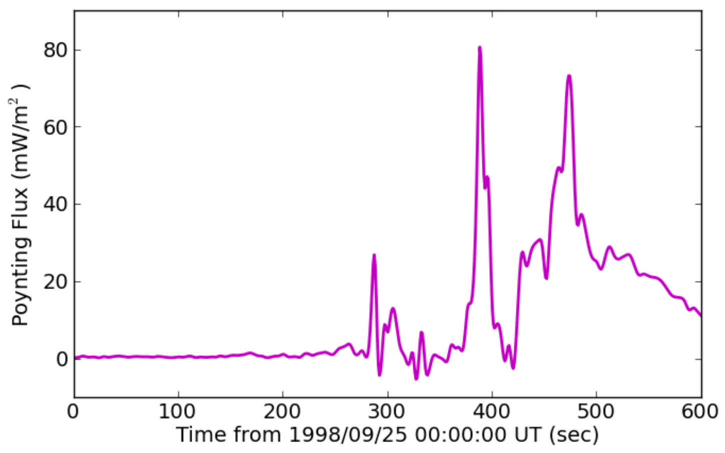


Figure 5.7: Interpolated data of the Poynting flux calculated from E-field and B-field measurements from a portion of FAST orbit 8276.

the detectors.

Data are provided as frames for a given moment in time, or timestamp. Each flux density frame is a 2-D array of energy versus detector number. Because of the spacecraft's rotation, each energy measurement is taken at a slightly different angle, and so a pitch angle spectrum is provided with each frame to give the average pitch angle view for each detector during the time it was accumulating data at each energy.

The flux density $I(E, \theta)$ is averaged over either energy or pitch angle to provide pitch angle or energy spectra, respectively. Integrating over both dimensions provides energy or number flux in the spacecraft spin plane, with the $(\hat{z})$ direction aligned with the magnetic field, according to

$$J_z = \iint I(E, \theta)\, \mathrm{d}\Omega\, \mathrm{d}E \tag{5.3}$$

$$N_z = \iint \frac{I(E, \theta)}{E}\, \mathrm{d}\Omega\, \mathrm{d}E \tag{5.4}$$

where J_z refers to the energy flux, N_z refers to the number flux, and $\mathrm{d}\Omega$ is the usual spherical expression $\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi$. θ refers to the spin plane, or pitch, angle. Because the FAST detectors do not view the more than 10° in the direction along the spin axis, we assume isotropy in ϕ, or around the magnetic field lines. Allowing θ to run over the full 360° in the spin plane, we must introduce an extra factor of 1/2 to the integral, giving the expression

$$\mathrm{d}\Omega = \pi|\sin\theta|\cos\theta\;\mathrm{d}\theta \tag{5.5}$$

The fluxes are then calculated with

$$J_z = \int_0^{16kHz}\int_0^{2\pi} I(E,\theta)\cdot\pi|\sin\theta|\cos\theta\;\mathrm{d}\theta\,\mathrm{d}E \tag{5.6}$$

$$N_z = \int_0^{16kHz}\int_0^{2\pi} \frac{I(E,\theta)}{E}\cdot\pi|\sin\theta|\cos\theta\;\mathrm{d}\theta\,\mathrm{d}E \tag{5.7}$$

5.2.3.1 Particle Processing Tools

`FAST.get_flux(I, E, dE, theta)`

The `FAST.get_flux` function performs the integration to obtain flux versus time for a given orbit. Energy flux is obtained by passing an array of ones with the same dimension as `E` so that the division in equation 5.7 is equivalent to equation 5.6. Integration is done using the trapezoidal rule, and so the array `dE` contains the differences between each energy bin value.

An alternative function `FAST.get_flux2` is also provided. This method performs the same integration, but removes the spacecraft velocity from the measured energies. The difference in flux values obtained between the two methods is negligible, but is more noticeable for high populations of low energy ions. Even in these cases, the difference is no more than 10%. To reduce computation time, `FAST.get_flux` is used for obtaining the electron fluxes and `FAST.get_flux2` is used for the ion fluxes.

The `FAST.get_spectra(orbit, species)` function creates energy and pitch angle spectra from the FAST data for both ions and electrons. The function returns the two spectra as well as the average pitch angle for each detector. The function was built for diagnostic purposes, and is not used in the preparation of data for statistical analysis.

Only lower energy ions contribute to ion outflow, but the specific range of energies of outflowing ions can vary. As such, we cannot set a hard limit on what energies to include when measuring ion outflow. This limit was previously determined by viewing the ion energy spectrum and selecting an obvious limit by eye. For example, the ion spectrum in Figure 5.8 has an obvious division near 300 eV. The lower energy ions have negative flux values while the higher energy ions have positive flux values, indicating both outflowing and precipitaing ions[2]. An algorithm was developed to predict this division and was shown to give reasonable judgements on the separation of outflowing from precipitating ions compared to the spectra. This algorithm is found in the function `FAST.cutoff(orbit, maxE)`.

[2]Since at the northern cusp the magnetic field points downward, using the $0°$ mark aligned with $\vec{B}$ gives negative values for outflow and positive values for precipitation. The ion fluxes reported in the next chapter are multiplied by -1 to indicate outflow.

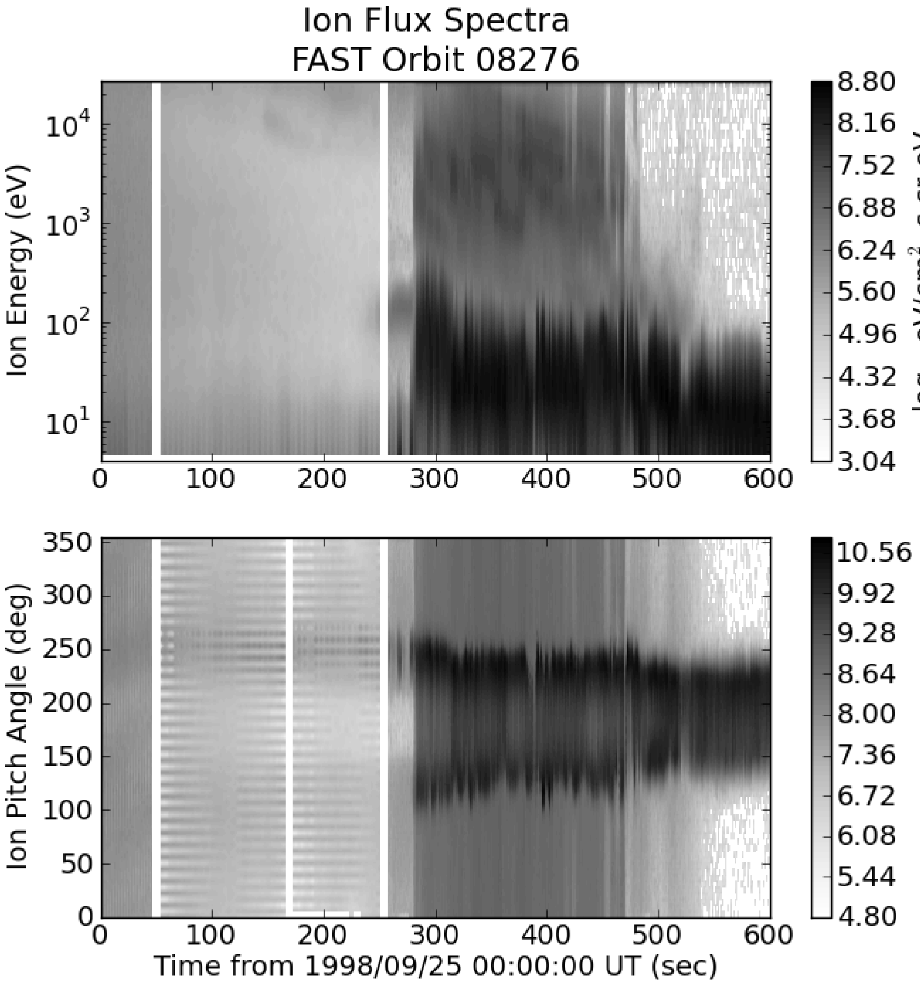


Figure 5.8: Energy and pitch angle spectra for ion fluxes measured in FAST orbit 8276. In the coordinate system used by FAST, 0° corresponds to the local magnetic field direction. Since at the northern cusp the field points downward, outflowing ions will be near the 180° direction, where we see a large flux in this example.

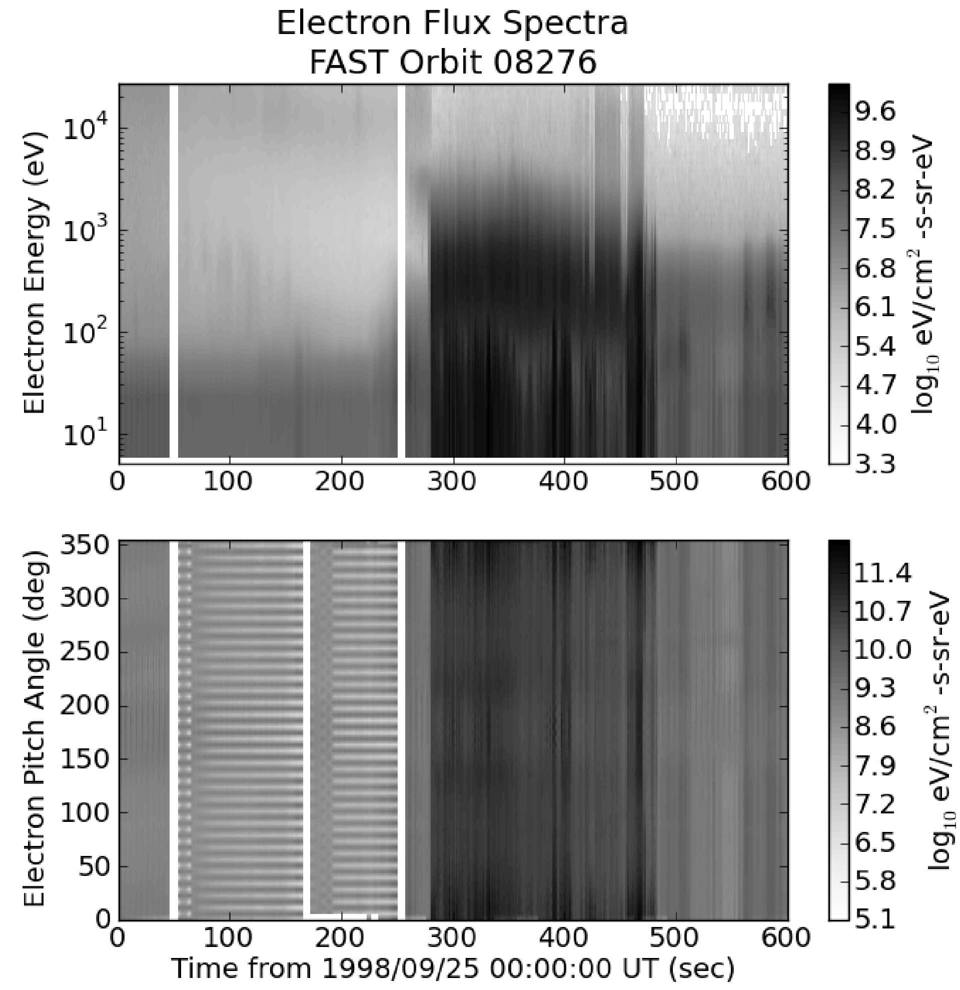


Figure 5.9: Energy and pitch angle spectra for electron fluxes measured in FAST orbit 8276. See Figure 5.8 for a description of the pitch angle variable.

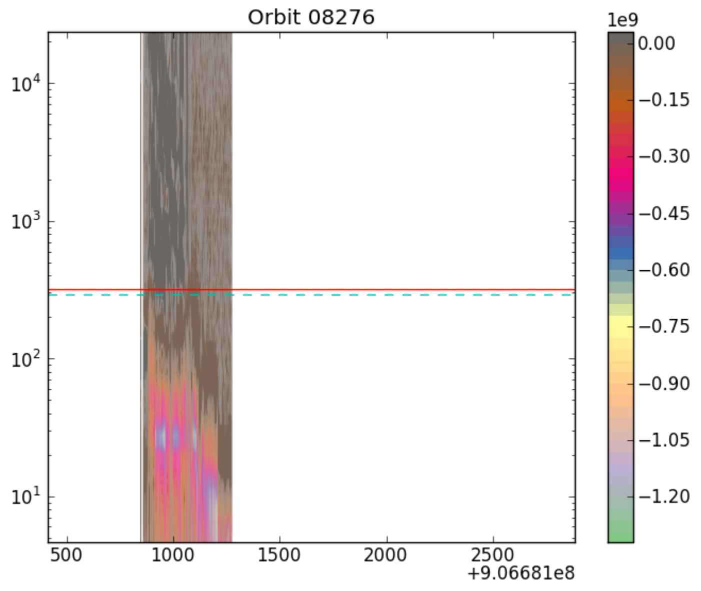


Figure 5.10: Example of the `FAST.cutoff` function results with the automated choice (solid red) nearly matching a visually selected choice (dashed cyan). The algorithm consistently matches cutoffs selected by hand.

The `FAST.cutoff` function calculates the ion number fluxes for each individual energy over a given orbit. Negative ion number fluxes indicate outflowing ions, so the cutoff is found by analyzing a number of characteristics of the data set. The energy steps where the mean number flux is negative and greater than 0.01% of the maximum outflowing number flux in magnitude are identified. These are further filtered to those showing at least 30% of the measurements over the orbit indicating outflow. The lesser of the highest energy matching these criteria and the specified maximum energy is selected as the cutoff for the analysis.

5.2.3.2 Electron Flux

`FASTget_EES(orbit, t_base, margin, subset)`

The electron number and energy fluxes are obtained with the `FAST.get_EES` function. This function is similar to the treatment described in section 5.2.2.2. Time endpoints are found, and the flux, energy, and pitch angle products are extracted from the CDF file. Fluxes are calculated for all energies above 50 eV. These data are not smoothed to 4 s intervals as the field data are, but are interpolated to the same time base. The time intervals between frames for the particle data are typically 0.3 s, and no smoothing is done in accordance with the treatment by Strangeway *et al.* [44]. This function returns both the energy flux and number flux over the interpolated time series.

5.2.3.3 Ion Flux

`FAST.get_IES(orbit, t_base, margin, subset)`

Ion number and energy flux are obtained in the same way as the electron fluxes, using `FAST.get_IES`. The only difference in this routine, aside from using `ies` data, is that energies are limited to above 4 eV and below the value obtained from `FAST.cutoff`. An arbitrary maximum limit of 500 eV is used for the cutoff value. The energy limits are returned along with the energy and number fluxes for reference.

A function `FAST.get_IES2` was written to obtain ion fluxes with corrections for

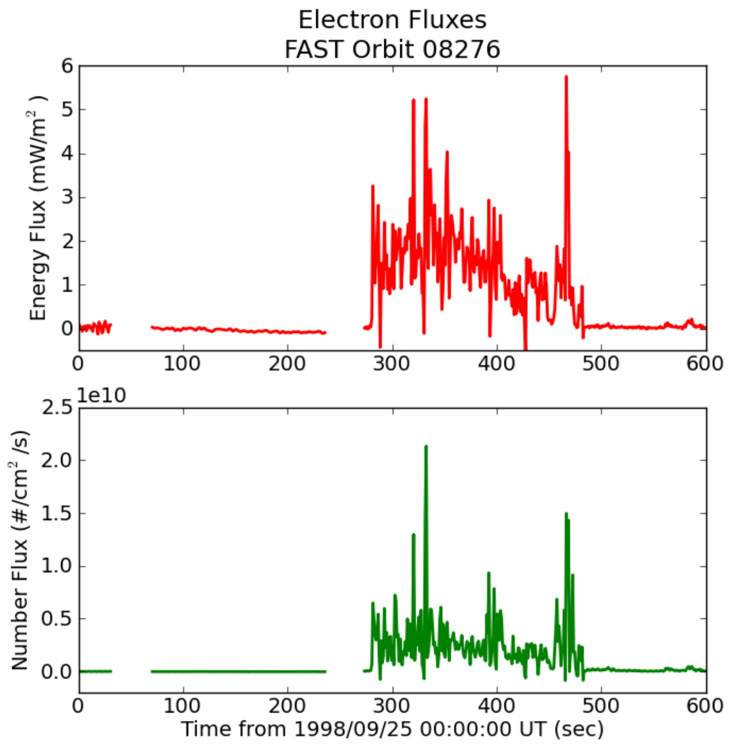


Figure 5.11: Electron energy and number fluxes from a portion of FAST orbit 8276.

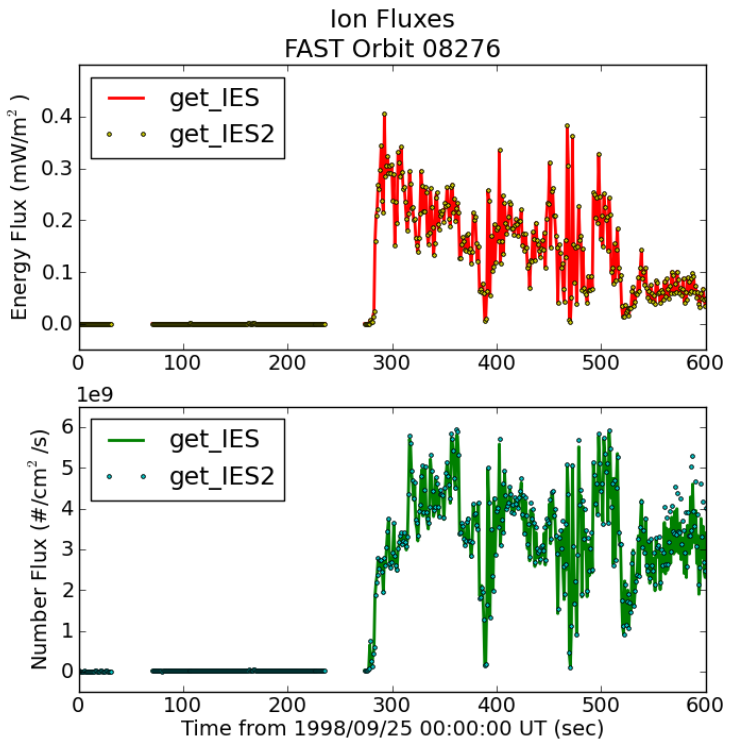


Figure 5.12: Ion energy and number fluxes from a portion of FAST orbit 8276. Fluxes obtained using satellite velocity corrections are overlayed for comparison. The velocity correction is only noticeable in situations where the energies of the ions are low. Note that in the last 100 s of this plot the energy flux falls by a factor of 4, and the number flux differs by ∼10% in the same time period.

the spacecraft motion using `FAST.get_flux2`. It is a little slower than `FAST.get_IES`, but does provide better accuracy for low energy ions (see Fig. 5.12).

5.2.3.4 Using FAST.py

The functions in `FAST.py` can be used individually after importing into a Python shell, or can be used in an automated fashion by making use of the function `processFAST(orbits, getelf)`. This function uses all of the other functions to create a data set including each data type interpolated to a common time base. The results are saved to the disk as a pickle.

`processFAST` expects a list of orbits to process. The CDF files for each data type should be saved to individual directories named by orbit, and the `basedir` variable should point to the directory where these directories are located. If a single orbit is to be processed it must still be passed as a list of length 1 (eg. `orbits=['08276']`). The `getelf` parameter is an optional boolean parameter to specify whether ELF data is available for the orbit. It is set as `True` by default.

Once run, `processFAST` will extract the orbit parameter data and create a flag array `CUSPF` that simply marks times where the orbit is over the cusp, as judged by being between invariant latitudes of 70 and 80°. Field and particle data are then processed, and a dictionary is created for the pickle. This dictionary has two other dictionaries as its elements. The `'data'` dictionary holds the actual data being saved, while the `'meta'` dictionary contains metadata pertinent to each element in the `'data'` dictionary, such as units and the software versions used in creating the data. The pickles are named by orbit and include information on the software version and Python version compatible with the pickle. Each pickle is saved within the same directory as its corresponding CDF files. The pickles can be loaded individually into Python as necessary for future processing.

The `processFAST` function runs for about 30 min. for each orbit in the list on a typical desk-top computer.

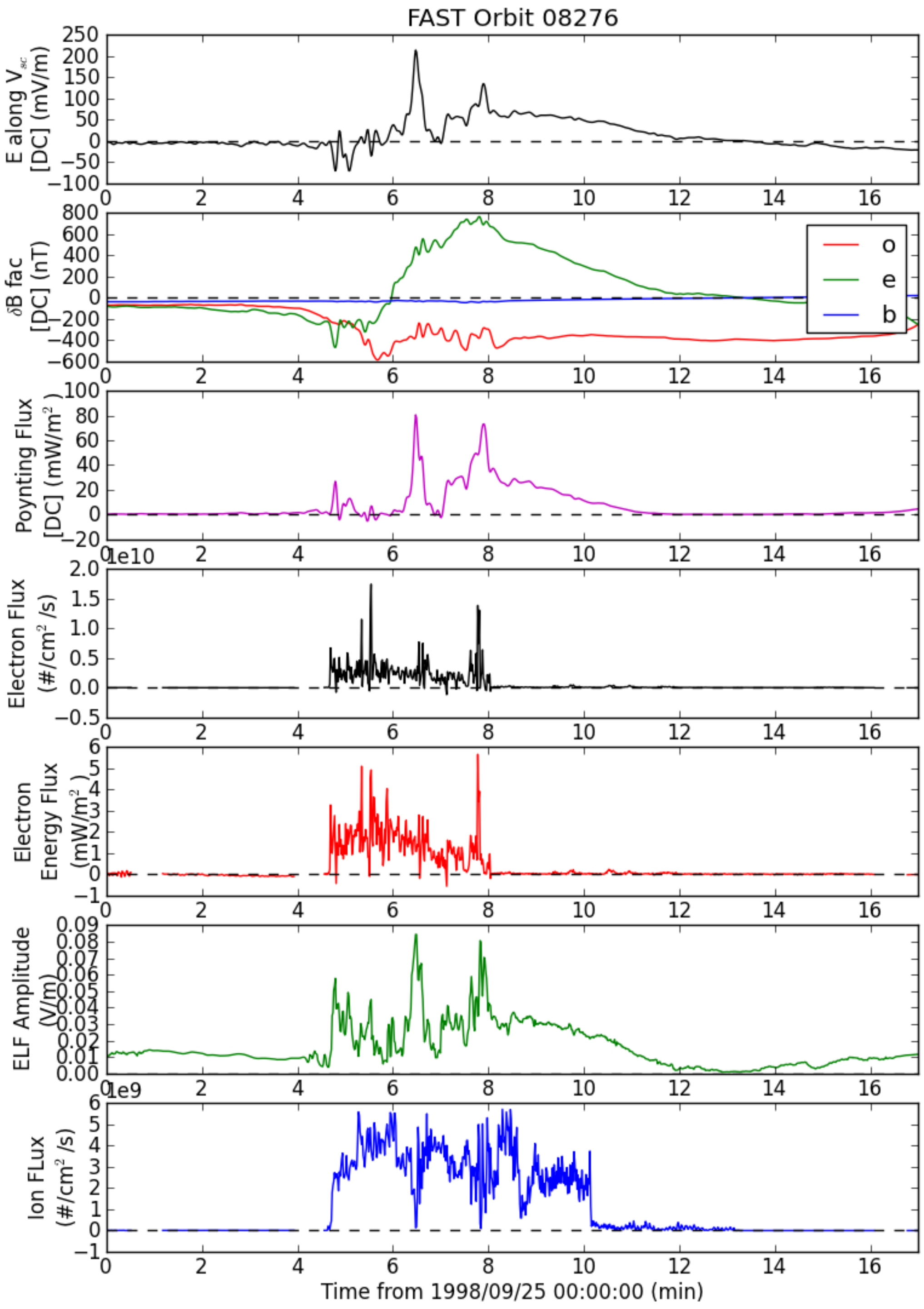


Figure 5.13: Summary of data from FAST orbit 8276 made using the routines in FAST.py to process the data. The panels show, from top to bottom, the electric field along the spacecraft trajectory, the three components of the magnetic field deviation, the Poynting flux, the electron number flux, the electron energy flux, the ELF wave amplitude, and the ion number flux. This figure is analogous to Fig. 3 in Strangeway 2005 [44].

5.2.4 FAST Statistical Analysis

This and the following section rely on the two Python programs `FAPlot.py` and `FAST_stats.py` (see Appendix E.1.2 and E.1.3 respectively). Before any statistical analysis can be done for the FAST data, the pickles returned by the function `FAST.processFAST` need to be updated to add an array identifying periods of ion outflow to be included in the study. While an automated method of doing this was sought, the number of changes from one orbit to another in the way ion outflow is identified proved too difficult, and this task remains to be done by hand. Much of the preparation to identify these periods can be automated, and so a function was created to be combined with user input as a means of identifying appropriate data for inclusion.

`FAPlot.update_pickle(orbits)`

The `FAPlot.update_pickle` function makes identifying periods of ion outflow simple. For each orbit in the list passed to the function, an ion pitch-angle spectrum is created using `FAST.get_spectra`. The spectrum is displayed with lines showing the invariant latitude (divided by two for convenience) and magnetic local time for reference. The code then pauses and asks for input to identify the starting and stopping indicies of data to be included. The plot is displayed against index number, making it simple to identify the desired points. Lines are displayed at the input indicies to verify the selection. Multiple segments of the data can be included. When all desired segments have been identified, an array is marked with ones for all

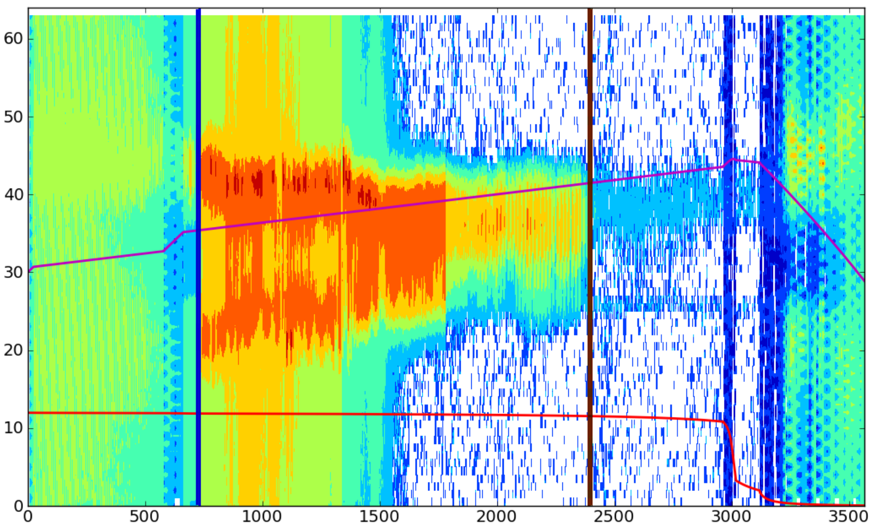


Figure 5.14: Example showing the spectrum for selecting ion outflow times from FAST orbit 8276. The two vertical lines correspond to the period selected. All data between these lines will be flagged as points to include in the statistics. In this study, data were not limited to only those regions where strong ion outflow is observed, as seen in this example.

points to be included. The array is then put into the dictionaries from the pickle, and the pickle is resaved with a modified name.

Periods to be included are primarily identified by including data showing strong ion conics, such as that seen in Figure 5.14. A difference in this study from the Strangeway study is that data from a larger time window were included in each orbit. Some orbits did not have strong ion conic signatures, but portions within the dayside cusp were included for a more comprehensive coverage of ionospheric conditions in the study.

`FAPlot.py` also includes a `basedir` variable to be used in the same manner as that in `FAST.py`. The `filesuff` variable should be set to match the suffix of the filenames for the pickles being processed.

`FAST_stats.get_averages(orbits, has_elf, alt_scale, length)`

Once the pickles have been updated, averages can be calculated. Averages are found using the `FAST_stats.get_averages` function, that provides a way to automatically collect averages from the FAST data with optional altitude scaling and period selection for the averages. By default, the function includes ELF data, does not do altitude scaling, and averages over the entire orbit. If desired, an integer number can be passed for the length parameter, allowing specific time intervals for the averages. (Eg. `length=60` would do averages over 60 points, or 60 s in the case of this study. See Figure 5.15.)

This function calls the `FAST_stats.get_nep(orbit, alt_scale)` function to generate the electron precipitation density, given by equation 4.2. When altitude scaling is enabled, all values are scaled to a standard 1000 km altitude assuming the flux scales as $\sqrt{B}$, or equivalently as $r^{-\frac{3}{2}}$.

When a collection of averages is obtained, linear regression is used to find linear fit parameters between pairs of data types. The 1-D linear regression routine in `scipy.stats` is used, which returns slope and intercept, the r-value correlation coefficient, the p-value test, and the standard error. Fits are made to the base-10 logarithm of each data type. As such, the slope indicates a power-law scaling value between energy inputs and the values to which they are regressed (in the case of the FAST data, the ion outflow).

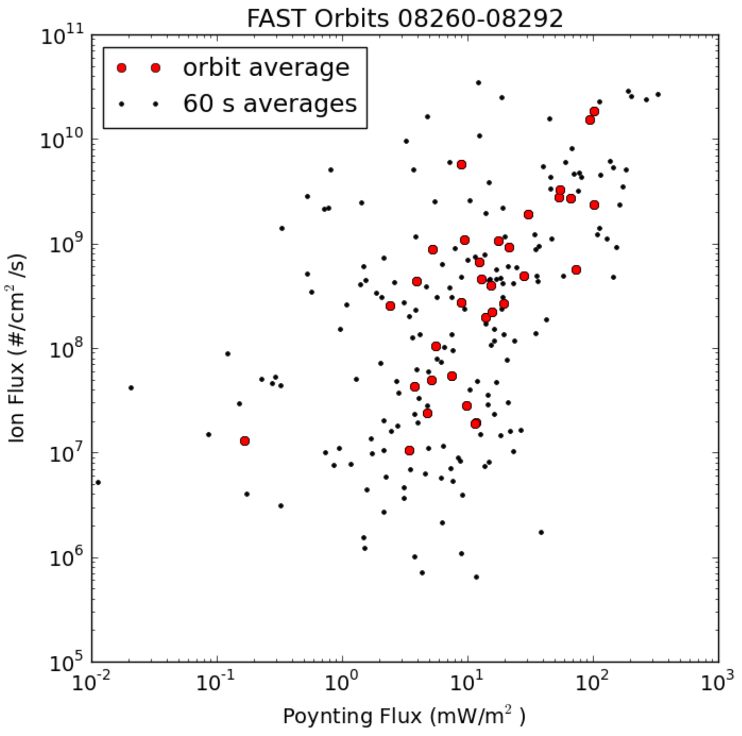


Figure 5.15: Plot showing orbital averages and 60 s averages for Poynting flux and ion number flux. Averaging over an entire orbit does give a good representation of shorter time averages, as the orbit averages maintain the general trend seen in the shorter averages. While there is more scatter in the shorter average data, a linear regression to the orbital average data is a good representation of the shorter average data.

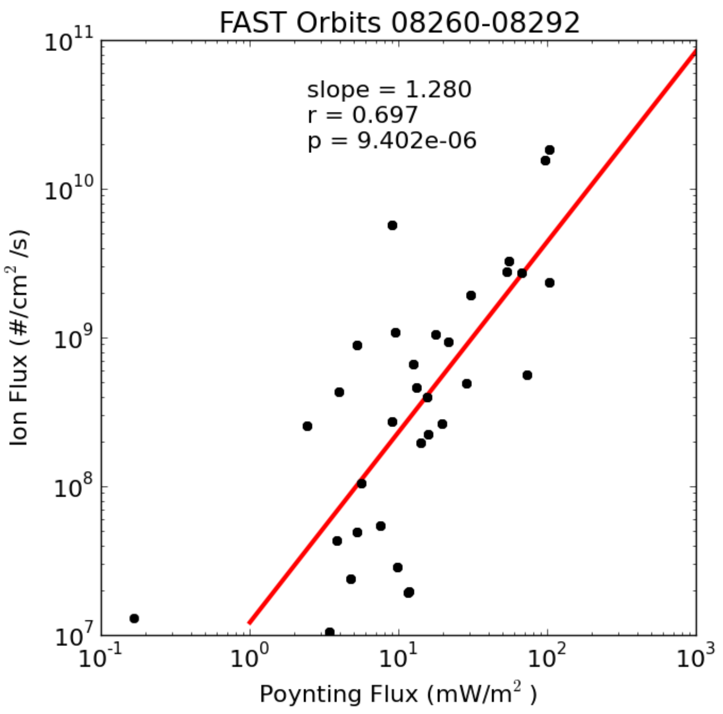


Figure 5.16: Linear Regression of FAST data from orbits 8260 to 8292. Each point is an average over an entire orbit. The regression parameters are similar to the values obtained by Strangeway for the same orbits.

5.2.5 FAST Data Visualization

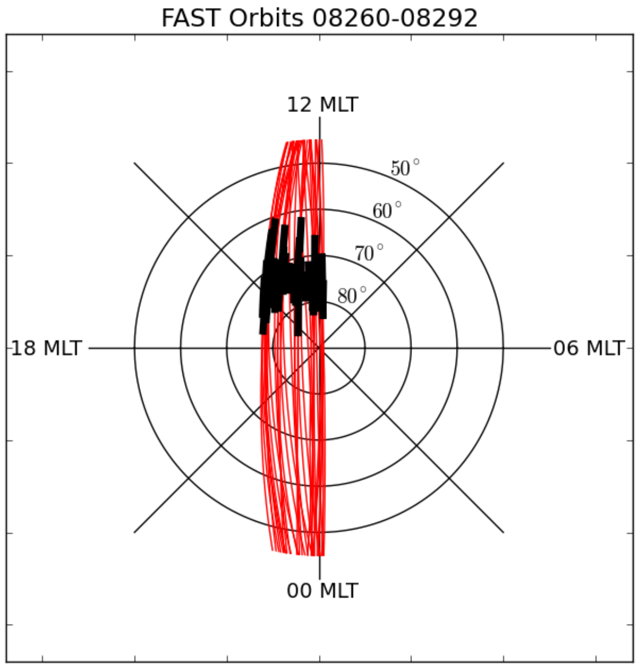
`FAPlot.gen_pole_plot(orbits, ofp)`

Two sets of code were developed to assist in examining the results of the data processing and statistics. In `FAPlot.py`, a polar plot of the orbit paths with the regions chosen for inclusion is useful to identify that the region of interest is, in fact, the dayside cusp (see Fig. 5.17). This plot is similar to one presented by Strangeway, and a comparison between them highlights the difference in the way the included data for the statistical averages were chosen. Our data inclusion is more comprehensive, and thus covers larger portions of each orbit than in Strangeway *et al.* [44].

The plot can be easily generated using the function `FAPlot.gen_pole_plot`. This function is passed a list of orbits with an optional parameter to plot only the portions where outflow is observed as chosen in the `FAPlot.update_pickle` routine. The plot in Fig. 5.17 was generated by running this function once with `ofp=False` and again with `ofp=True`.

`FAST_stats.get_plot(x, y)`

The `FAST_stats.get_plot` function is used to generate a log-log plot of the two data types passed as `x` and `y`, with a linear fit and the key statistical parameters displayed as seen in Figure 5.16. The function first removes any data points with nan values, then generates the scatter plot with linear fit. Note that since ion outflow has a negative flux along the magnetic field line, the ion outflow data are given a



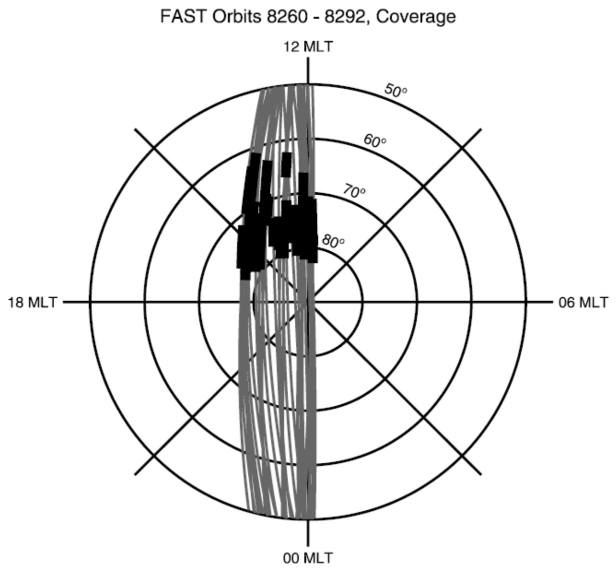


Figure 5.17: Plots showing portions of the orbits used in the statistics presented in Fig. 5.16 (above) and in the Strangeway study [44] (below).

sign change to generate this plot, as well as in all other plots showing the ion number flux.

`FAPlot.gen_summary_plot(orbit)`

Finally, a simple routine in `FAPlot.gen_summary_plot` is provided to produce a plot similar to the one seen in Figure 5.13 for any specified orbit. This function asks for user input to select the starting time in the format `YYYY,M,D,h,m,s`. Leading 0's should be left out in this input.

5.2.6 CHAMP Data Processing

CHAMP data types are available at the ISDC website in different formats. The types pertinent to this study include the STAR accelerometer data (acc) and the rapid science orbit data (rso). Both of these data types are found in an ASCII text format detailed in the CHAMP Data Format [64] and the CHAMP Orbit Format [65].

Because the CHAMP data come in an ASCII format rather than CDF, a different coding scheme was necessary to parse the data for use. CHAMP data are also organized by day rather than by orbit, and each file includes data from multiple orbits. The scheme developed for processing the CHAMP data uses an interactive menu to identify what data types are available in a given file and select which data types will be extracted.

Care must be given to applying the following code to any data from CHAMP, as some files are formatted with slight changes. While the changes are unimportant when looking at the text file directly, they make any automated data extraction scheme impossible. The code presented here was tested and worked successfully for the time periods used in this study (both in 2002). Some changes may be necessary to generalize it to other time periods.

5.2.6.1 Orbit Data

`champ.pso_extract()`

CHAMP orbit parameters come in three types: rapid science orbits (RSO), predicted science orbits (PDO), and post-processed science orbits (PSO). Because the availability of the post-processed orbit data is not complete, the rapid science orbit data are used. The RSO data come in 14-hour files. The filename is descriptive of the data set, with the format `CH-OG-3-RSO+CTS-CHA_<YYYY>_<DOY>_<HH>.dat`. In this filename, CTS refers to the coordinate system used, the Conventional Terrestrial System, or GEO. CHA specifies the data is for the CHAMP satellite. `<YYYY>`, `<DOY>`, and `<HH>` give the year, day of year, and hour for the start of the file. In the time periods studied, data were available in two files per day, starting at 10:00 AM and 10:00 PM UT.

The CHAMP orbit data format provides time, position, velocity, and attitude data in addition to flags indicating the conditions for the satellite at that particular time (eg. over land/water, in the Earth's shadow, etc.). Time is provided as the number of days from the J2000.0 epoch (noon on 1 January 2000) and the fraction of a day in microseconds. Data are provided for 30 s intervals. Position is given as X, Y, and Z in mm for the GEO coordinate system. Velocity is given for X, Y, and Z in 10^{-7} m/s. A column is provided in the orbit data format for the neutral gas density, however it will only be non-zero in PSO data, and for the time periods studied was not used even there.

Orbit data are provided with one line of data per time stamp. The locations of each data type are specified and consistent, allowing easy parsing of the data. The `champ.pso_extract` function automates this process with input from the user to select which data types should be extracted. Because the data format is the same for both PSO and RSO data, the routine can be used for either type.

5.2.6.2 Accelerometer Data

`champ.acc_extract()`

CHAMP accelerometer data also come in ASCII format, but are handled differently than the orbit data. The filename follows a similar convention, where `CH-OG-2-ACC+<YYYY>_<DOY>_<HH>.n.dat` indicates the data type (accelerometer), the year, day of year, and starting hour as before. The numerical value n indicates the revision number of the software that produced the data file. In general, the most recent revision should be used when multiple files are available.

The ACC files include important information in the header. Most of the information is unnecessary for the data itself, but calibration constants are included that are necessary to obtain the correct values from the accelerometer data. These

Char #	Description
1–6	Time tag (10**-1 d since J2000.0)
7–17	Time tag (10**-6 s since 0 hours)
18–29	X coordinate of position (10**-3 m)
30–41	Y coordinate of position (10**-3 m)
42–53	Z coordinate of position (10**-3 m)
54–65	X coordinate of velocity (10**-7 m/s)
66–77	Y coordinate of velocity (10**-7 m/s)
78–89	Z coordinate of velocity (10**-7 m/s)
90–96	Roll angle (10**-3 deg)
97–103	Pitch angle (10**-3 deg)
104–110	Yaw angle (10**-3 deg)
111–115	Neutral gas density (10**-16 g/cm**3)
116	Maneuver flag (M = yes, else blank)
117	Land/water flag (L = Land, W = Water)
118	Ascending/descending arc flag (A = ascending, D = descending)
119	Eclipse flag (E = satellite in Earth's shadow, else blank)

Table 5.2: Summary of the CHAMP orbit data format.

	CHAMP Spacecraft Coordinate System
Origin	Spacecraft center of gravity
$\hat{X}_{sc}$	Aligned with the long side of the spacecraft towards the boom, in nominal attitude pointing in flight direction (roll axis)
$\hat{Y}_{sc}$	Forming a right-handed system with $\hat{X}_{sc}$ and $\hat{Z}_{sc}$ (pitch axis)
$\hat{Z}_{sc}$	Nadir pointing in nominal attitude (yaw axis)

Table 5.3: Description of the coordinate system used to reference the position of instruments relative to the CHAMP satellite. The first column lists the coordinate system component, and the second a description of how that component is defined. Instruments define their viewing direction relative to this coordinate system.

constants provide a scale and offset to the individual components of acceleration and should be applied to the data before use.

Accelerometer data are provided for 10 s intervals and come in an instrument fixed, inertial coordinate system as described in the CHAMP Reference Systems, Transformations and Standards [66]. The spacecraft coordinate system is body-fixed, with $\hat{X}_{sc}$ aligned with the long dimension of the spacecraft, nominally in the flight direction, $\hat{Z}_{sc}$ pointing in the nadir direction (or always toward the Earth), and $\hat{Y}_{sc}$ completing a right-handed triad. The instrument coordinate system is aligned such that $\hat{X}_{acc}$ is anti-parallel to $\hat{Z}_{sc}$, $\hat{Y}_{acc}$ is parallel to $\hat{X}_{sc}$, and $\hat{Z}_{acc}$ is anti-parallel to $\hat{Y}_{sc}$. In this coordinate system, $\hat{Y}_{acc}$ points along the trajectory and gives the acceleration felt by the spacecraft from atmospheric drag. This component provides the desired density information, and is the only component used in this study.

	STAR Accelerometer Reference Coordinate System
Origin	Accelerometer proof-mass centre = (nominal) Spacecraft Center of Gravity
$\hat{X}_{acc}$	Anti-parallel to $\hat{Z}_{sc}$ (less sensitive axis)
$\hat{Y}_{acc}$	Parallel to $\hat{X}_{sc}$
$\hat{Z}_{acc}$	Anti-parallel to $\hat{Y}_{sc}$
Φ	Rotation about $\hat{X}_{acc}$
Θ	Rotation about $\hat{Y}_{acc}$
Ψ	Rotation about $\hat{Z}_{acc}$

Table 5.4: Description of the coordinate system used by the STAR accelerometer. Note that the vector components are defined relative to the spacecraft coordinate system given in Table 5.3.

ACC data is given in chunks, with one line giving the time stamp and subsequent lines prior to the next time stamp providing other data types. Not every data type appears in every time stamp, and so parsing the file to extract the data requires watching for key words to identify when the different data types are used. When a data type is missing for a given time stamp, a `nan` value is inserted to ensure the array lengths are equal. The `champ.acc_extract` function extracts the calibration data and the data values for each time stamp with user input to identify which data types should be kept.

5.2.6.3 Using champ.py

`champ.extract_data()`

The `champ.py` code is written as an interactive menu for selecting the data to be extracted. The `champ.filename` variable should be set to the name of the file to be extracted. `champ.extract_data` asks the user to specify the file type given in `champ.filename`. A menu comes up listing the data types found in the file. By default, all data are included in the extraction, but the menu allows the user to select only the data types desired. When the list of included data is deemed satisfactory, the function calls the appropriate extraction function for the specific data file type.

`champ.process_CHAMP_orb(directory, days)`

In addition to the individual file handling routines, there are two routines provided in `champ.py` to automate the extraction of data for a given list of days.

`champ.process_CHAMP_orb` handles the orbit data processing for multiple files. The directory where files are located is specified, and a list of days (given in the format `'yyyy_ddd'`) is passed to the function for processing. The individual files are located that contain data for each day to be processed, and data is extracted from each using the generalized `champ.extract_data` routine. The time stamps are converted to UNIX time with `champ.time_convert(d, s)`, and the position and velocity components are converted to SM coordinates using the routines provided in `cstrans2.py`. The data for each day are put into a dictionary and saved as a pickle. Note that while an entire day of data is spanned by three files as provided by GFZ Potsdam, starting with the later file of the previous day, the pickles merge the data from the different files to create one single file per day, starting at 00:00 UT. For this study, only the time, position, and velocity data are used. Other data included in the CHAMP files may be selected out in the `champ.extract_data` menu.

`champ.process_CHAMP_acc(directory, days)`

The `champ.process_CHAMP_acc` function handles the accelerometer data processing for multiple files. As before, the proper data files are identified, though for ACC data only one file exists per day. The CHAMP data revision number is specified in the code, and this must be modified if that particular revision is not available for that day. The menu is used to select the desired data types. The extracted time stamps are converted to UNIX time and the calibration data are applied to the $\hat{Y}_{acc}$ component acceleration data. The time and calibrated decceleration are added to the dictionary, which is then saved to a pickle. For this study, only the time and calibrated decceleration are necessary.

5.2.7 CHAMP Statistical Analysis

The choice of data is the primary difficulty in comparing CHAMP data to FAST data. The two satellites have different orbits and do not make measurements in the same region at the same time. Even when the orbital planes are aligned, the two satellites have very different altitudes, with FAST at 350–4175 km, and CHAMP near 400 km. By selecting orbits that have nearly aligned orbital planes, however, it is possible to choose portions of data for analysis if we assume the altitude difference is unimportant.

`CHAMP_stats.create_pickle(days)`

To be used together, the extracted CHAMP data are interpolated onto a common time base, using the standard 10 s intervals provided by the accelerometer data. The acceleration data must also be interpolated to allow for missing time stamps. The `CHAMP_stats.create_pickle` function creates an interpolation time base and provides interpolated data for invariant position (ie. invariant latitude, magnetic local time, and altitude), velocity, and decceleration.

`CHAMP_stats.find_upwelling(fa_orbit, fa_basedir)`

Identifying the proper data for inclusion in the study is done using the function `CHAMP_stats.find_upwelling`. For a specified orbit of the FAST satellite (including the directory where the orbit data is stored), the invariant latitude of both the FAST and CHAMP satellites are plotted with the CHAMP decceleration data. The FAST data are examined to identify the range of invariant latitudes on the dayside of the magnetosphere that have been pre-selected for inclusion in the statistics. (Ideally, this corresponds to the region where ion outflow is observed by FAST.) The corresponding regions (dayside invariant latitudes in the same range as those identified in the FAST outflow data) are marked in the CHAMP accelerometer data, as is the time during which FAST was passing through the region. From this information, the CHAMP data to include in the statistical analysis is selected. An example of this process is shown in Figure 6.12, and discussed further in section 6.5.1.

The CHAMP data that are selected correspond with the region of interest that exhibits neutral upwelling, evidenced by an increase in decceleration, an indication of an increase in atmospheric density. The increased decceleration region is then marked for interpolation, using data of an equal time window to that selected on either side of the spike. The difference between the decceleration increase and the interpolation from the baseline density from that observed on either side is used to provide a measure of the amount of neutral upwelling occurring for the given orbit. The baseline decceleration was found using a linear interpolation routine. The mean and peak value of the difference is returned by the function.

5.2.8 CHAMP Data Visualization

When averages are obtained for a list of corresponding FAST orbits, the functions in `FAST_stats.py` and `FAPlot.py` can be used to examine the data. No further routines specific to CHAMP were used in this study.

5.3 Assumptions

Any analysis is subject to uncertainty due to the underlying assumptions made in the analysis. Space missions require many simplifying assumptions in order to work with data, and care was taken to ensure the underlying assumptions of this study were reasonable and justified. This section will describe the key assumptions in this study.

5.3.1 Assumptions in Field Quantities

Electric field data are assumed to be an accurate representation of electric field strengths along the spacecraft trajectory. This assumption is valid for FAST data prior to Sep 2001, however due to a faulty probe on the spacecraft the assumption is less certain for the 2002 data analyzed. Examination of the 2002 electric field data showed that the data had been compromised by the faulty probe as evidenced by large oscillations in the data. However, the averages may still show the general trends as evidenced by the results presented in the next chapter.

Magnetic field variations are calculated by finding the difference between measurements made with the FAST magnetometers and the IGRF-7 1995 modeled magnetic field. We assume that the configuration (strength and position) of Earth's magnetic field has not changed significantly since 1995. Since IGRF-7 was only designed to be valid until the year 2000, this assumption is a little weaker for the 2002 data. However, since all FAST data (including recent data) are processed with this model, IGRF-7 was used to remain consistent with the data set.

5.3.2 Assumptions in Particle Quantities

The particle detectors on FAST have a limited view perpendicular to the spin plane of the spacecraft. Flux values are found under the assumption that the ionospheric plasma is isotropic in this direction. This assumption is typically made in plasma analyzers, and seems to be a reasonable simplification in measuring plasma fluxes in the ionosphere, where much of the plasma is attached to the magnetic field.

5.3.3 Assumptions in Neutral Density

While the CHAMP spacecraft decceleration is proportional to the neutral density, it is also proportional to the square of the spacecraft velocity. The CHAMP orbit is nearly circular, and so the velocity changes only slightly over the course of an orbit. The resulting change in the corrected decceleration accounting for a difference in orbit velocity is negligible, and the decceleration data are assumed to scale only with density.

The primary goal of the STAR accelerometer was to measure gravitational fluctuations. We also assume that any gravitational fluctuation in this direction is negligible compared to the atmospheric drag. Since we only use the decceleration component in the direction of motion of the spacecraft, which is nearly perpendicular to the vector toward the center of the Earth, this assumption is reasonable.

5.3.4 Assumptions in Calculations

In calculating Poynting flux, only the $\hat{e}$ component of magnetic field is used. The desired quantity of S_z should also have a contribution from the $\hat{o}$ component, but deviations from the model in this direction are small compared to those in the $\hat{e}$ direction. This component is assumed to always be negligible in comparsion, and is left out in the calculation of S_z. In addition, the calculation of the cross-product assumes that δB_e is perpendicular to $E_{V_{sc}}$. The spacecraft trajectory over the cusp is nearly perpendicular to the local magnetic field, and so this assumption is also reasonable.

Precipitating electron density is calculated using the formula derived by Strangeway *et al.* [44]. This formula, as given in Equation 4.2, was derived through dimensional analysis, and is assumed to be an accurate representation of the density of electrons precipitating into the cusp.

Chapter 6

Statistical Analysis

Statistical analysis of the data was done on three different time periods. The first of these periods centered on the 25 Sep 1998 event studied by Strangeway, but includes orbits 8227 through 8325, spanning nearly 219 hours of time between 20 Sep and 29 Sep 1998. Five of these orbits were not included due to the software not detecting any significant ion outflow (viz. orbits 8236, 8295, 8302, 8305, and 8308). In this time period, the results show the consistency of the relationships found in the Strangeway study, even with more data over a broader range of outflow conditions included.

The second time period studied is centered on an event identified on 6 July 2002. This event (as seen in Fig. 6.1) is similar in nature to that observed in the 1998 event, and there is excellent conjunction between the FAST and CHAMP satellites, both crossing the northern cusp aligned with the 09:00–21:00 magnetic meridian (see Fig. 6.2). Data from FAST orbits 23387 through 23418 are included, spanning a little less than 70 hours between 4 Jul and 7 Jul 2002. Data from all orbits in this time period were used. The results from this time period are compared to the Strangeway results. This event is also used to examine neutral upwelling.

The third time period studied is a collection of 60 orbits during the first week of February 2002. These data were used because a different type of outflow signature

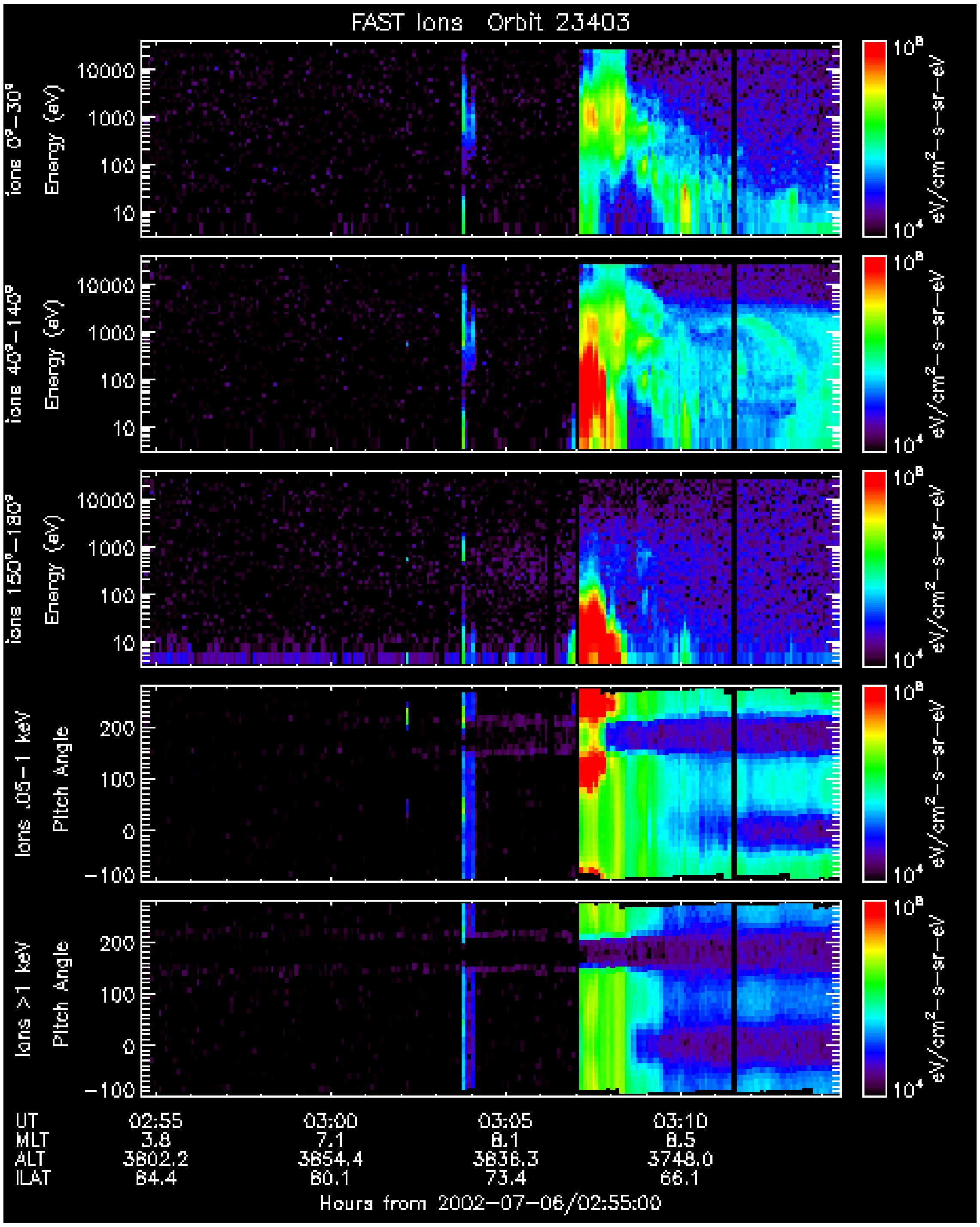


Figure 6.1: Pitch-angle spectrogram summary of the observed event in July 2002. This event has characteristics similar to the Sep 1998 event, showing strong ion conics in the low-energy pitch-angle spectrum, and large fluxes of outflowing ions in the energy spectra. (See Fig. 4.3 for comparison.)

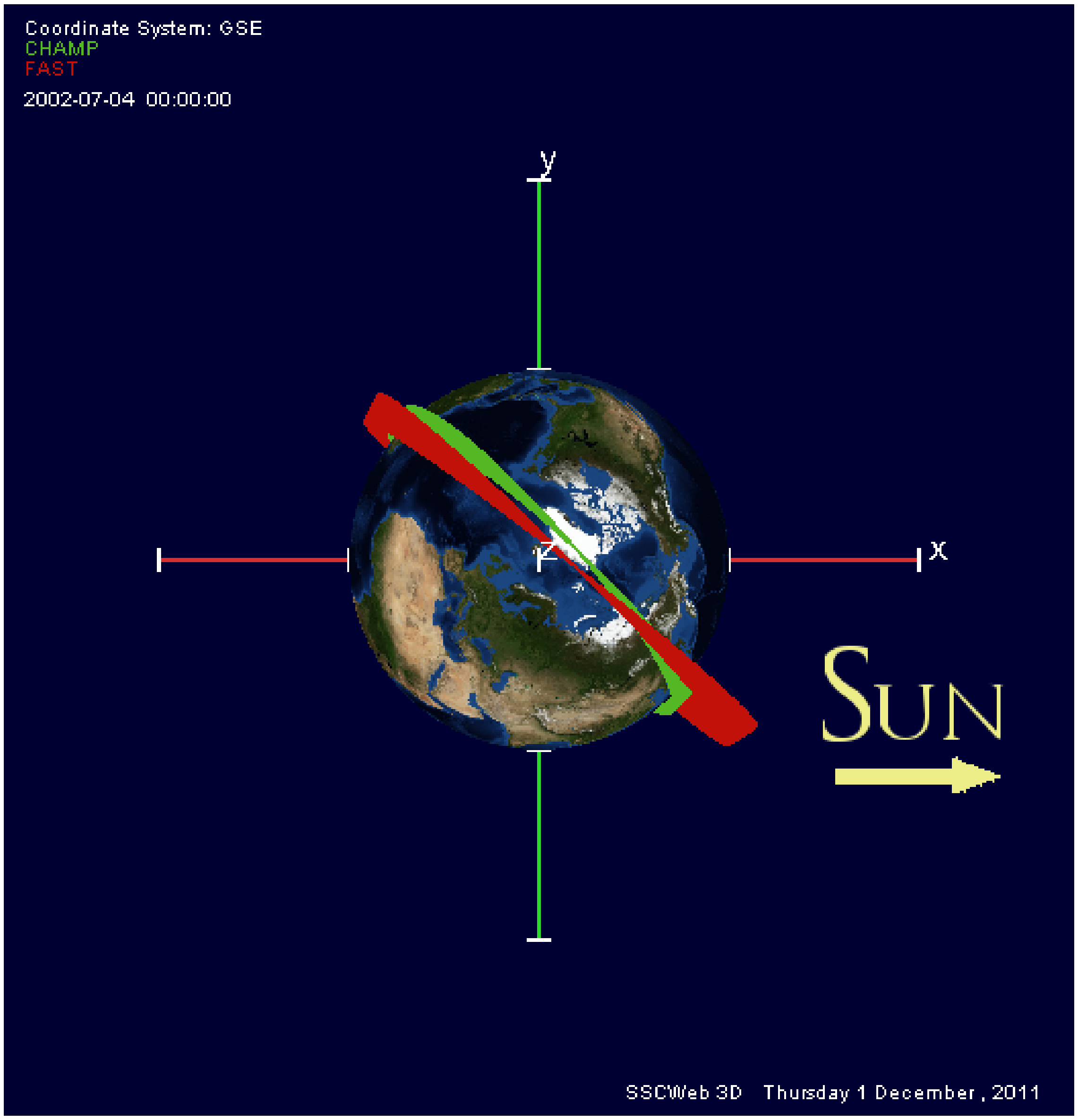


Figure 6.2: Orbital coverage of the time period studied in July 2002. The FAST and CHAMP satellites are aligned to within 5° and oriented along the 09:00–21:00 magnetic meridian.

was seen fairly constantly throughout the month of February in this year, and also because of good alignment between the FAST and CHAMP satellites along the noon–midnight magnetic meridian (see Fig.s 6.3 and 6.4). Because of the weaker signals in this month, not every orbit contributed to this data set. FAST orbits 21688 through 21747 were analyzed, with 21694, 21696, 21701, 21703, 21707, 21718, 21733, 21742, and 21744 not contributing. The overall time period spans just over 130 hours from 1 Feb through 6 Feb 2002. This event examines the correlations at a different time and under different ionospheric conditions. This time period is also used to examine neutral upwelling.

All data sets are evaluated at a scaled altitude of 1000 km to account for variations in altitude. The scaling process used is described in Section 5.2.4. In each of the following plots, the data presented are scaled to this altitude.

6.1 Time Average Consistency

One of the assumptions to be examined is the validity of correlating average values when taken over the entire selected time period of each orbit as opposed to multiple points averaged over shorter time periods. In each orbit, the measured values can vary over a wide range. For example, in orbit 8276, the Poynting Flux measurements can range between 0.1 and 100 mW/m^2. The upper portion of Figure 6.5 shows the change in orbital average by averaging over data that has been smoothed further to 10 through 90 s intervals (by 10 s increments) compared to the average over the entire orbit from the 1 s data used in the study. The average value

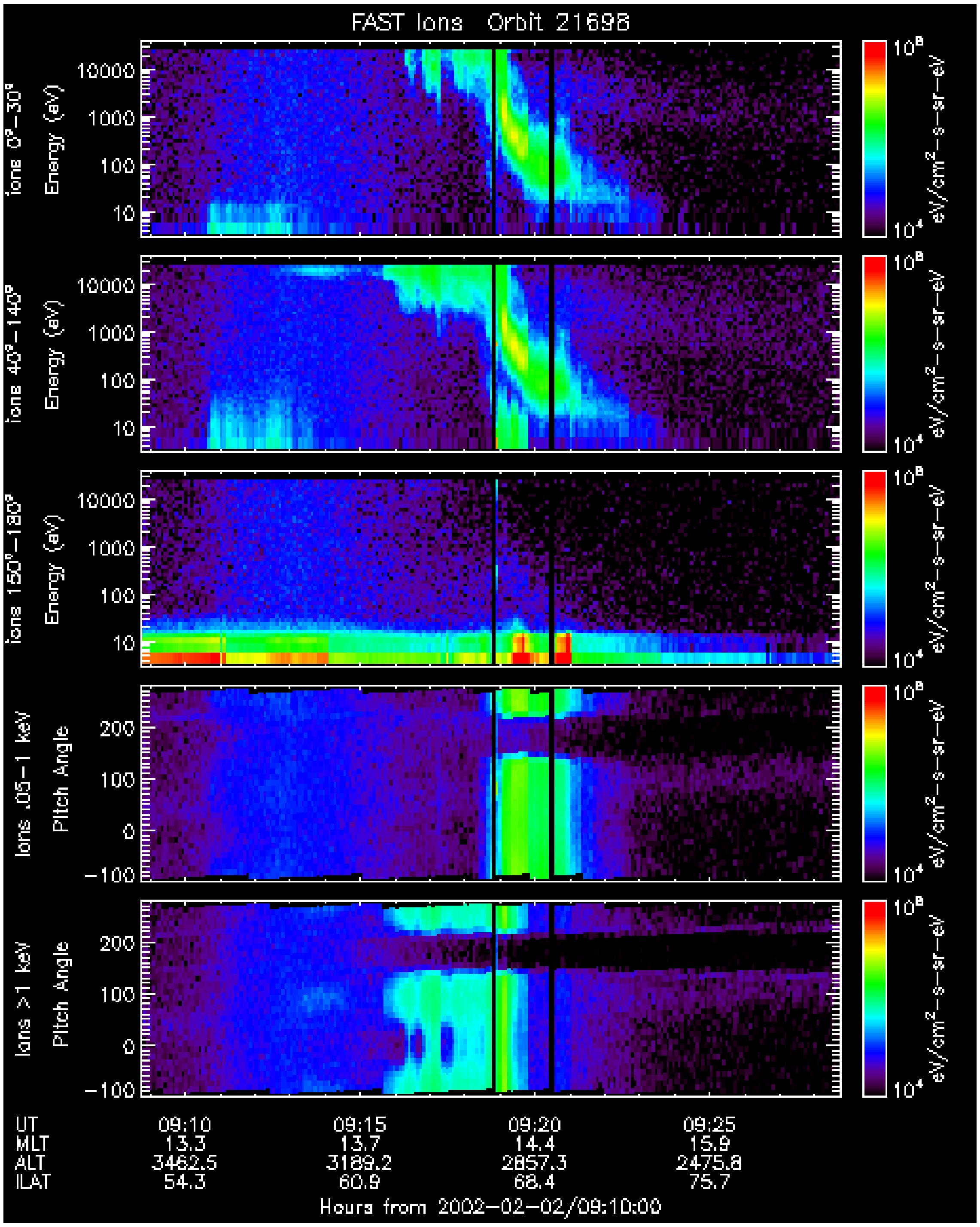


Figure 6.3: Pitch-angle spectrogram summary of the observed event in February 2002. While the ion conics in the low-energy pitch-angle spectrum are weak, strong ion outflow are seen at energies less than 10 eV. The energy spectra of down-flowing ions at high energies are seen in many passes of the FAST satellite during this month.

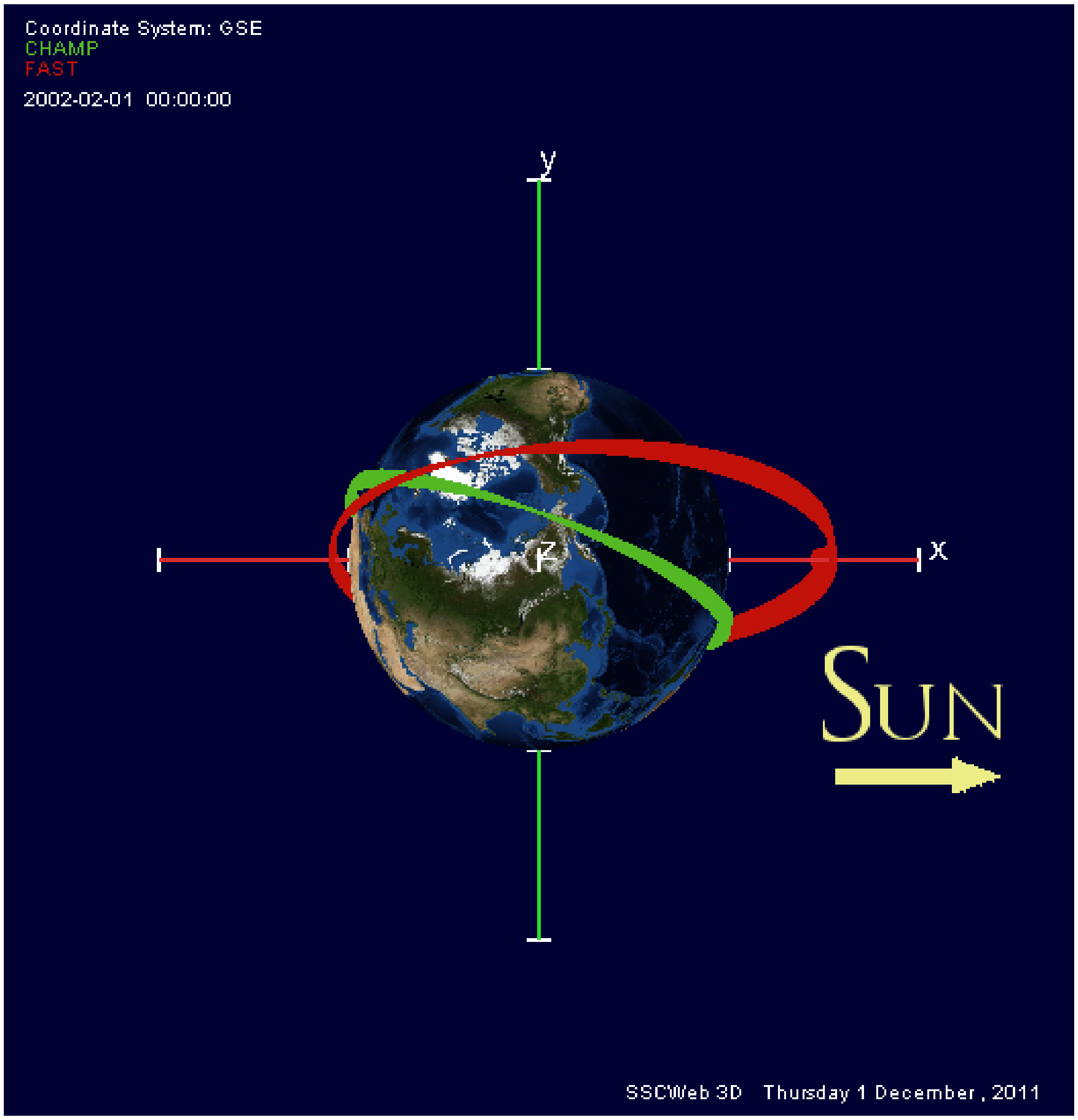


Figure 6.4: Orbital covereage of the time period studied in February 2002. The FAST and CHAMP satellites are aligned to within 10°, with an orientation along the 12:00–00:00 magnetic meridian.

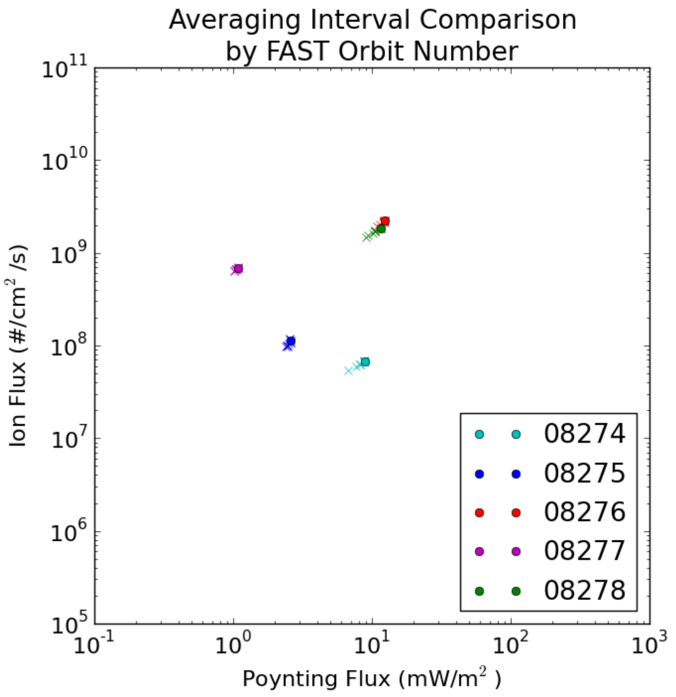
for the orbit does not change significantly with the added smoothing, and so the orbital average is a good representation of the orbit as a whole.

In addition, an examination of the orbital average as it compares to 10 s and 60 s averages for each orbit is presented in the lower portion of Figure 6.5. This figure shows that the use of an average value over the entire orbit as opposed to a number of shorter averages for each orbit is an adequate representation of the ionospheric conditions measured. While beyond the scope of this work, it is interesting to note the difference in orbit 8275 (before the onset of the storm) varies greatly in ion outflow with reduced ion number fluxes, while orbits 8276, 8277, and 8278 (all during the storm) show greater variation in Poynting flux and an order of magnitude or greater increase in ion number flux. A logarithmic average may also be appropriate to use here, but to a simple linear average was used in this study.

It is also worth examining whether the trends established by the orbital averages are consistent with individual data. Figure 6.6 shows that the orbital average points are indeed consistent with trends seen using averages of 10 s and 60 s. Shorter time intervals broaden the distribution in the scatter plot, but the general trend is still clearly visible.

6.2 Poynting Flux

The results of the statistical correlation between Poynting flux and ion outflow for the Sep 1998 data are shown in Fig. 6.7. We see good agreement between the Strangeway result and this result, especially when considering only the same orbit



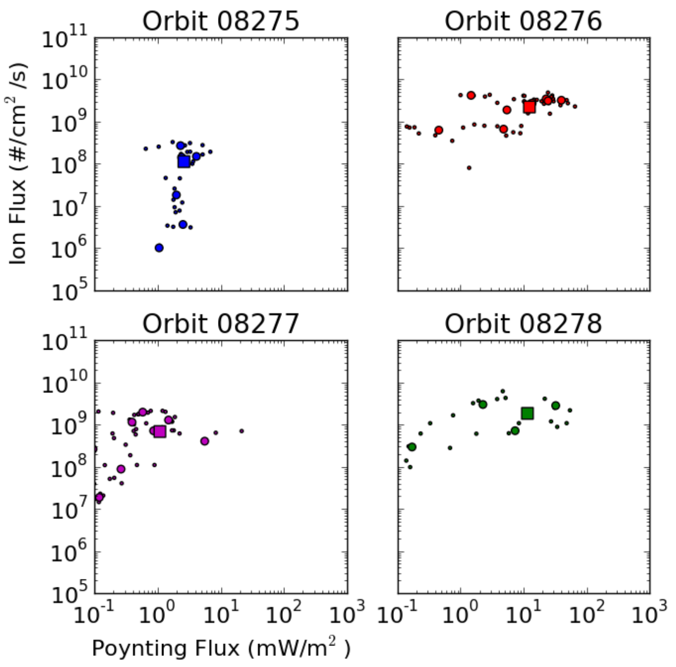


Figure 6.5: Plots showing the effect of using shorter time intervals for averaging. The overall change is not significant from using the average over the entire orbit rather than averaging over shorter intervals. Below are scatter plots using all data for incremental average intervals. The orbital average represents the overall data well.

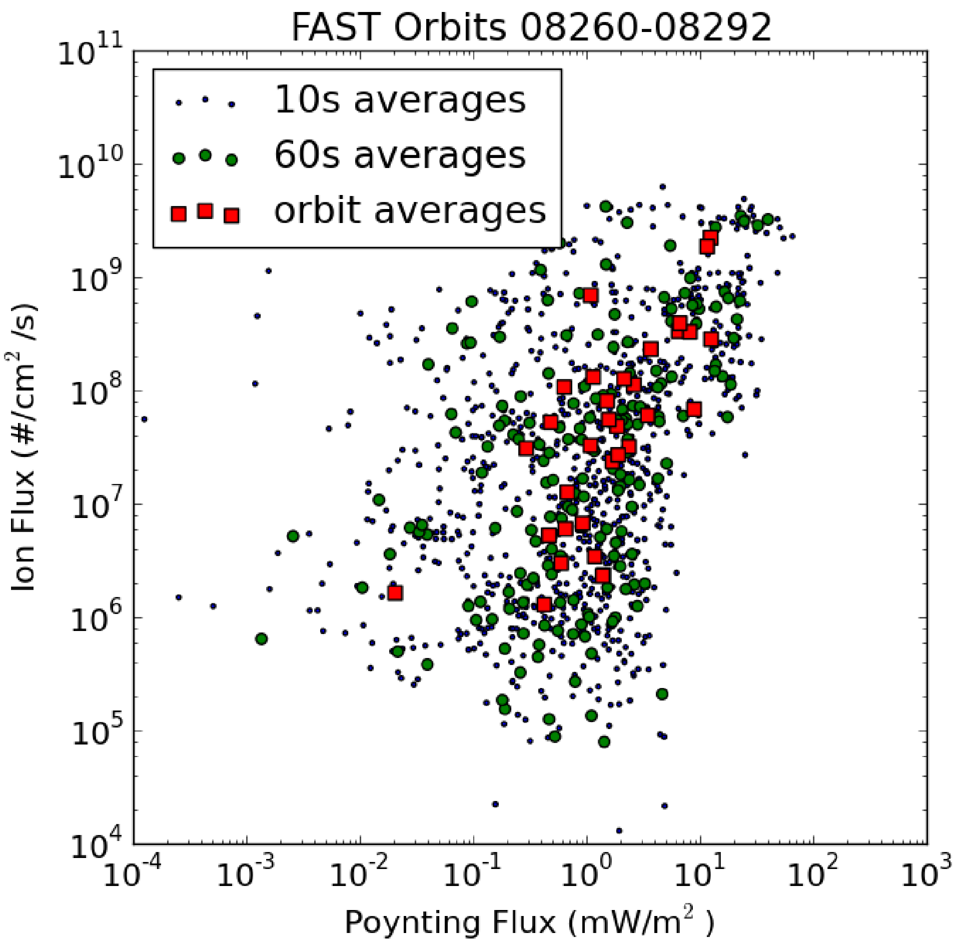


Figure 6.6: Data from FAST orbits 8260–8292 by averaging interval. The orbital averages are an accurate representation of the overall trend seen in the shorter interval data.

set (FAST orbits 8260–8292). The parameters of the linear regression of these averages also agrees with the Strangeway result, supporting the general validity of the correlation between Poynting flux and ion outflow. Because the methods used to select data for inclusion in this study selected a broader range of data from each orbit, this agreement indicates that arguments against his findings based on selectivity of data are unlikely to be correct. The included data for this study cover a much longer period of time, extend beyond the impact of the studied event on both ends, and include a more larger portion of each orbit's data in the calculated averages. Even with all of these changes, the results are consistent, suggesting it is valid to describe a power law relation between the Poynting flux and the ion outflow at the northern dayside cusp. The observed power law exponent is approximately 1.2.

Electric field data from FAST are compromised in 2002, however the statistical analysis of this time period shows a correlation similar to that found in 1998, with a slope within 10% of the previous fit. The observed power law exponent in this data set is approximately 1.3. The shift observed in the data seen in Fig. 6.8 could be attributed to a strong attenuation of the measurement of the real fields observed in the damaged V5 probe. The fact that a correlation consistent with the 1998 period results may indicate that on average some of the signal may retain a portion of the actual electric fields observed by the FAST satellite in this time period and not just the faulty oscillations. This conclusion is supported by the correlation found for electron precipitation density to be presented shortly. Nevertheless, because of the degredation in data quality this result should not be taken as a confirmation of the

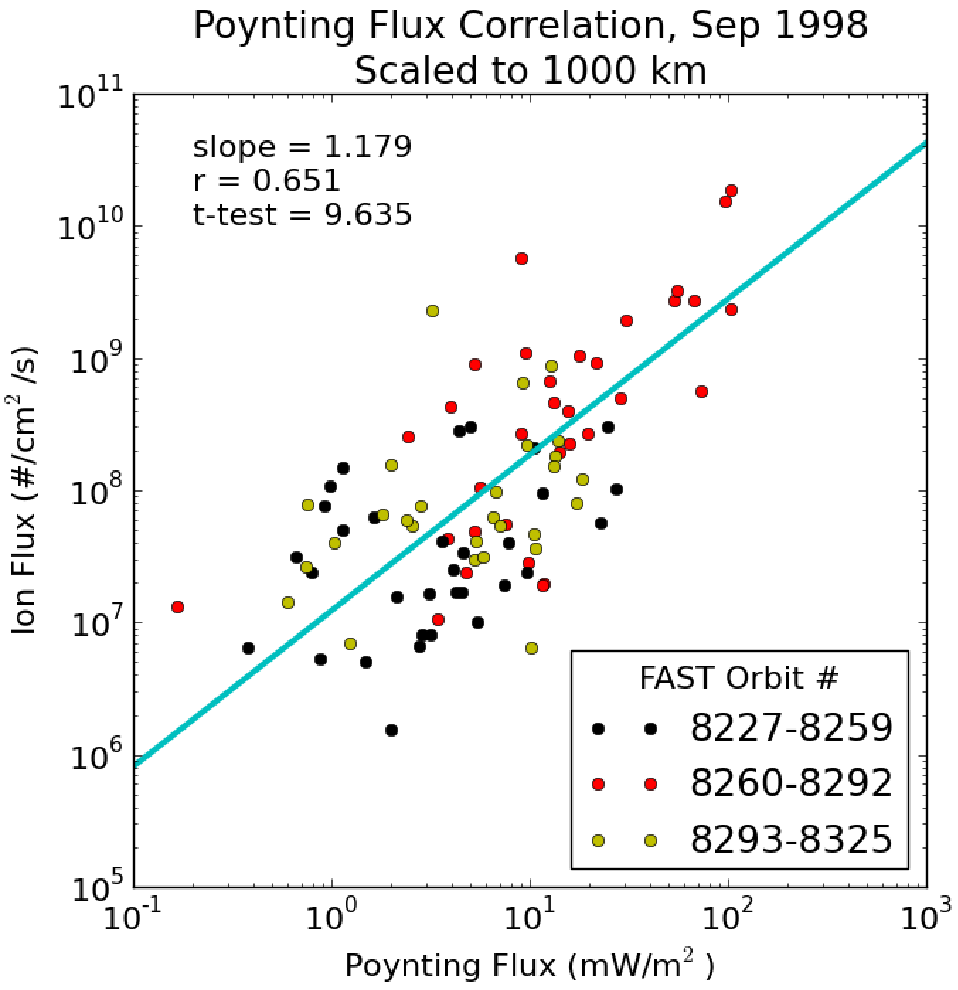


Figure 6.7: Poynting flux correlation for the orbits in September 1998. This and the subsequent plots all show the measured data scaled to a 1000 km altitude. See Sec. 5.2.4 for details on the altitude scaling.

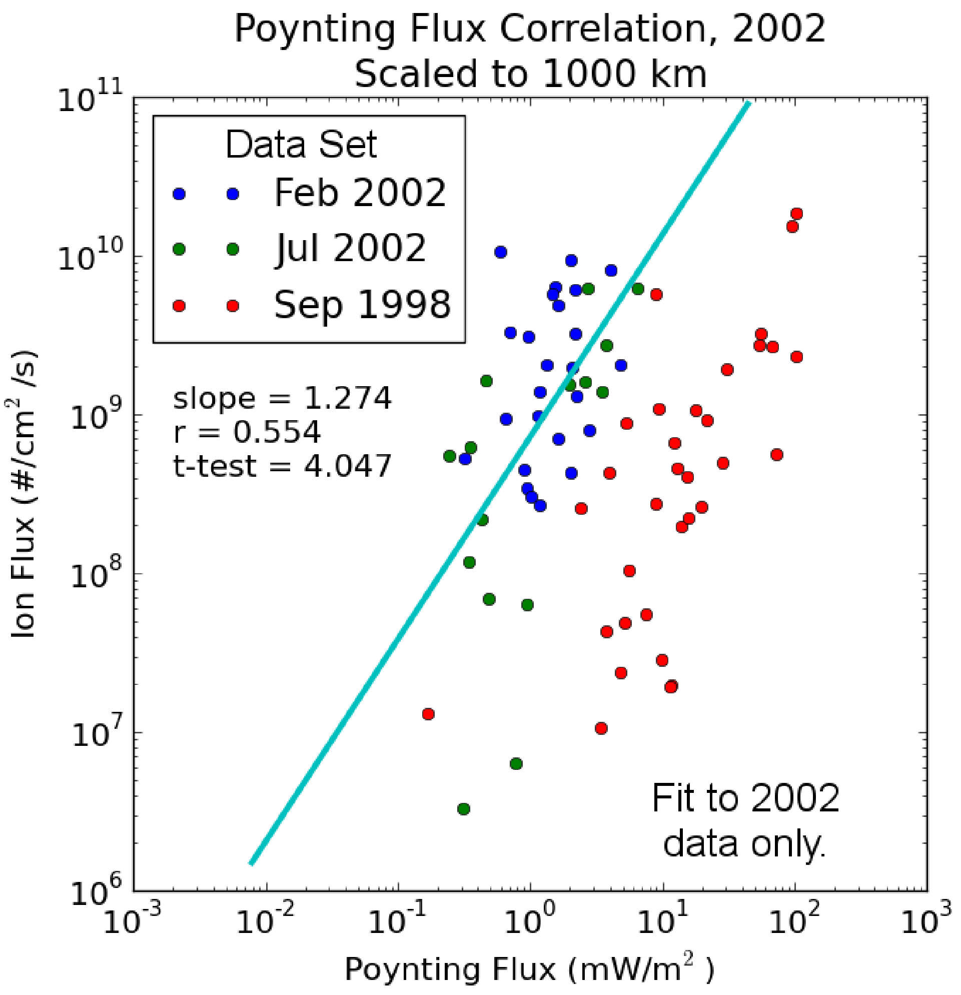


Figure 6.8: Poynting flux correlation for the orbits in February and July 2002.

Strangeway result.

6.3 Electron Precipitation

The results of the statistical correlation between electron precipitation density and ion outflow for the Sep 1998 data are shown in Fig. 6.9. As in the case of the Poynting flux statistics, we again see agreement with the Strangeway result. When

Data Set	*Slope*	*r-value*	*t-test*
Strangeway-2005	1.27 ± 0.45	0.72	5.80
Sep 1998	1.18 ± 0.24	0.65	9.64
All 2002	1.27 ± 0.63	0.55	4.05

Table 6.1: Summary of the statistical results for the Poynting flux correlation. Uncertainties are for a 95% confidence interval. The r-value gives the correlation coefficient, which is a measure of the linearity of the data. (Larger r-values are better.) The t-test is a statistical test describing how likely the result is to be significant. (Again, larger numbers are better.)

removing a few outlying data points for the purpose of obtaining a more descriptive fit, we find again that a power law description is appropriate, with an exponent of 2.1. While other factors play a role in ion outflow, electron precipitation density is consistently found to correlate with ion outflow.

The result of the electron precipitation density correlation in 2002 reveals a similar trend to that found in 1998 (again, with outlying data points removed in the linear regression). Particularly, the Jul 2002 data are found to agree with the Sep 1998 data, as can be seen in Fig. 6.10. The Feb 2002 data also agrees in the trend, showing a nearly identical slope to the regressions of the other data sets. An increase in the observed ion outflows in this month may indicate a difference between the types of events studied, but it is clear that the statistical relation between electron precipitation density and ion outflow is consistent over the times studied.

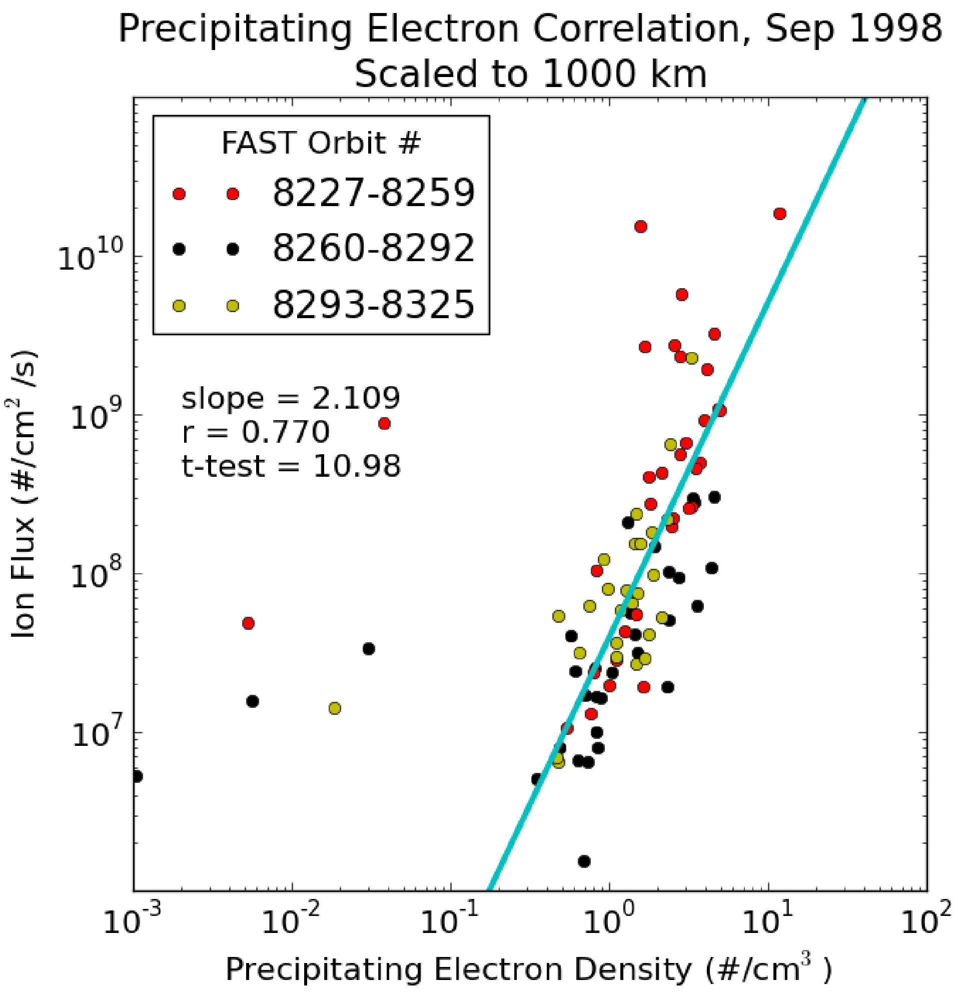


Figure 6.9: Electron precipitation density correlation for the orbits in September 1998.

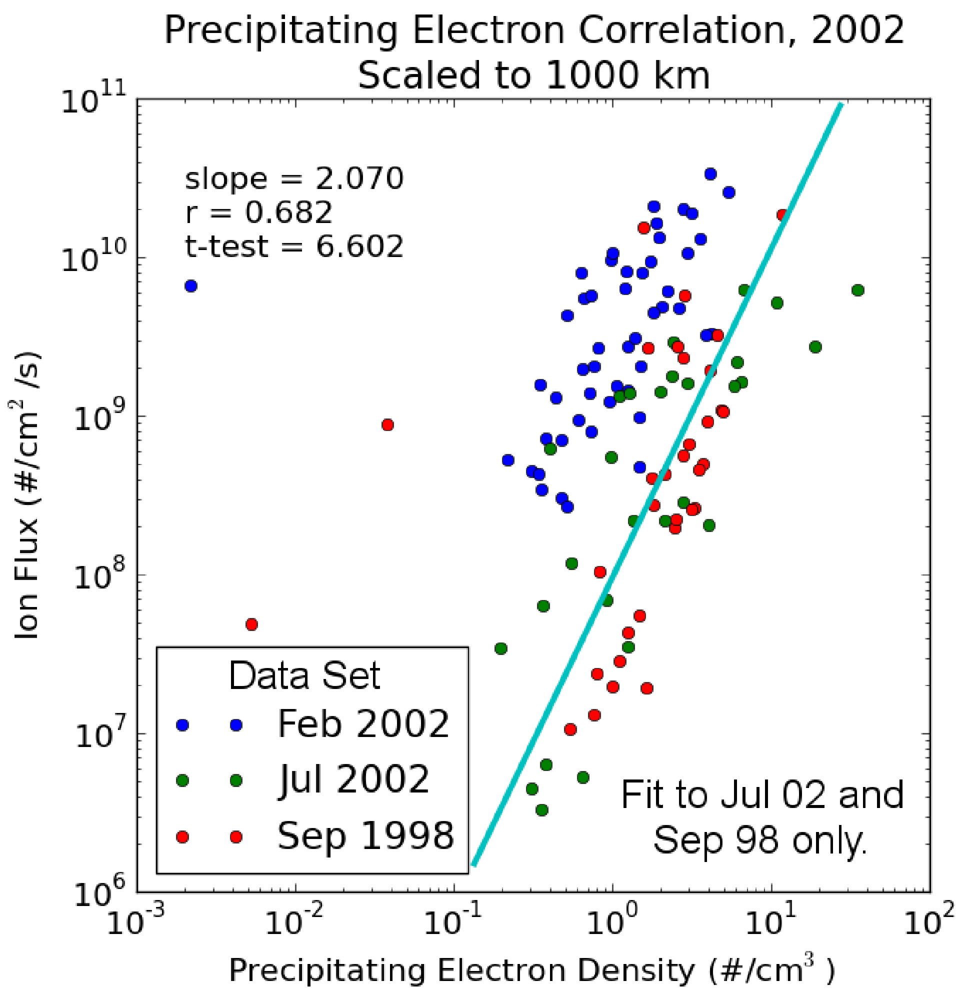


Figure 6.10: Electron precipitation density correlation for the orbits in February and July 2002.

Data Set	*Slope*	*r-value*	*t-test*
Strangeway-2005	2.20 ± 0.49	0.86	9.18
Sep 1998	2.11 ± 0.38	0.77	10.98
Feb 2002	1.41 ± 0.37	0.76	7.72
Jul 2002	1.83 ± 0.56	0.82	6.72
Jul 02 & Sep 98	2.07 ± 0.63	0.68	6.60

Table 6.2: Summary of the statistical results for the electron precipitation density correlations. Uncertainties are for a 95% confidence interval.

6.4 Cross Correlations

The cross-correlation found by Strangeway between the ionospheric energy inputs studied is also present in the data used in this study. The selected data from 2002 show a weaker cross-correlation between Poynting flux and electron precipitation density than found by Strangeway, due to a larger spread in the February data (see Fig. 6.11). The July data alone show a strong cross-correlation, with a slope of 1.18 and corresponding r-value of 0.73.

6.5 Neutral Upwelling

6.5.1 Visual Observations from Selection of CHAMP Data

Not every orbit of the CHAMP satellite sees evidence of neutral upwelling. Many factors go into whether a density enhancement is observed, but two observa-

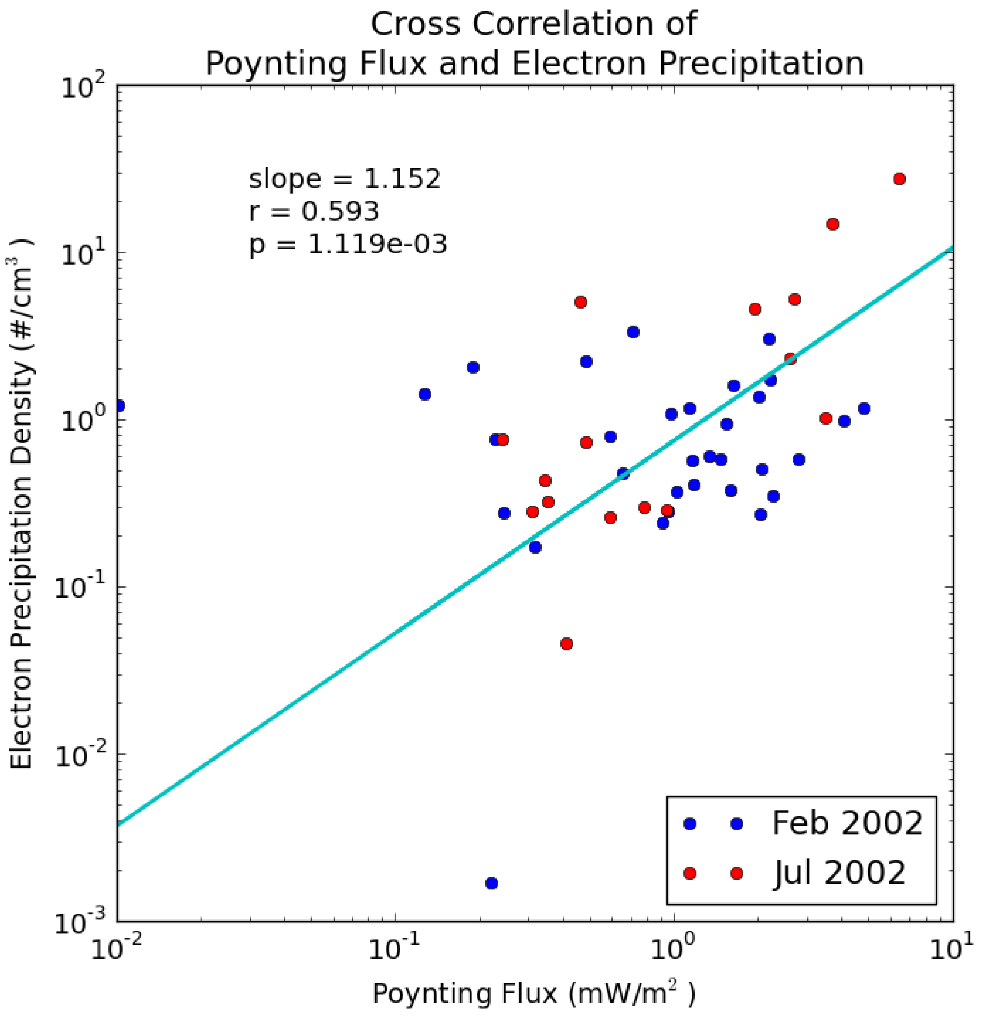


Figure 6.11: Cross correlation between Poynting flux and electron precipitation density for FAST data in 2002. Note that the July data alone is found to show a stronger correlation.

tions made while selecting data to include from the CHAMP accelerometer provide good insight to some of those factors. First, when an enhancement is seen in the data, it always occurs at or around the same range of invariant latitudes of the day-side magnetosphere where the FAST satellite observes ion outflow. This observation leads us to expect some sort of correlation at least in location. Second, on some orbits (see for example the data from FAST orbit 21688 in Fig. 6.12) CHAMP will pass over the cusp twice in the orbital period of FAST. There are cases where both passes observe neutral upwelling, and others where only one pass will show a density enhancement in the cusp. In the latter case, the second pass is always the one to show the enhancement, indicating there may be some time delay between the onset of an ion outflow event and the upwelling of the neutral gas.

Correlations were obtained for the various ionospheric energy inputs with the neutral density represented by the decceleration of the CHAMP satellite. The decceleration value used is the deviation from the baseline decceleration as interpolated from the orbital trends observed by CHAMP. Both the mean deviation and the peak deviation from the baseline were examined.

6.5.2 Poynting Flux

Examining the correlations for neutral density showed no correlation between Poynting flux and CHAMP decceleration (see Fig. 6.13). This result may be due to the compromised electric field data from FAST. Another possibility may be that Poynting flux is negligible during neutral upwelling events. Schlegel *et al.* found

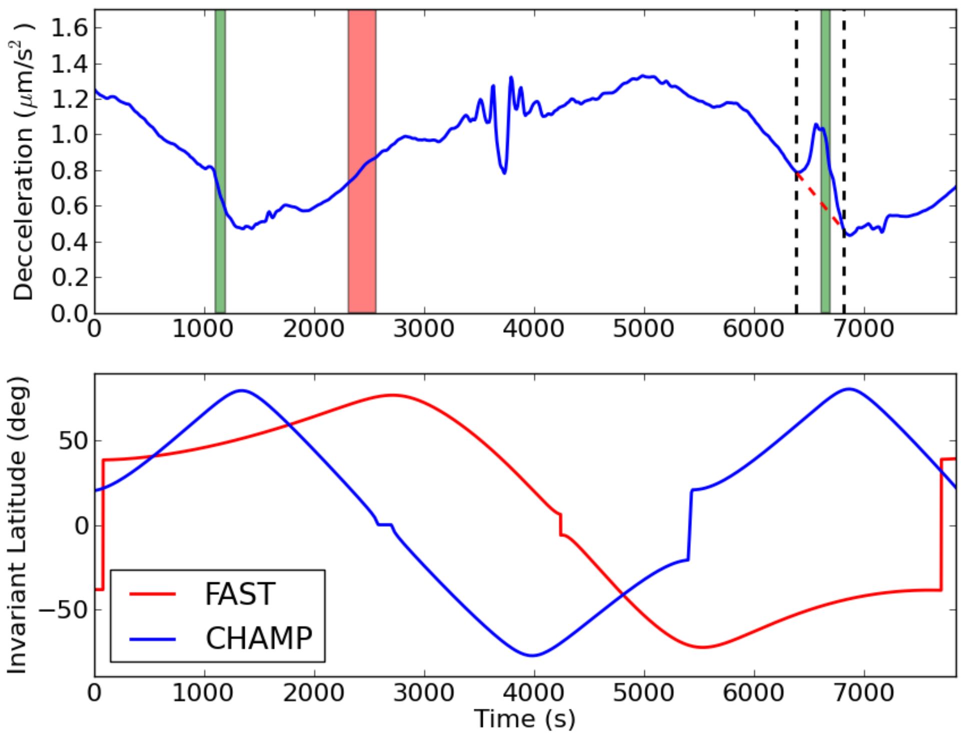


Figure 6.12: Sample plot showing how CHAMP data are selected for study using FAST orbit 21688. On the lower plot, invariant latitude for both satellites is plotted against time, indicating the synchronicity of their passing over the CUSP. On the upper plot, the CHAMP accelerometer data are plotted with indicators to assist selection. The green regions indicate portions of CHAMP data from the same latitude range of the dayside magnetosphere where ion outflow was observed by FAST. The red region indicates the time interval when FAST is passing over those latitudes. The black, vertical dashed lines indicate the portion of data selected for study, and the red dashed line below the selected data shows the baseline density interpolation for the selected region.

Data Set	*Slope*	*r-value*	*t-test*
Feb 2002	-0.15 ± 0.19	0.27	1.58
Jul 2002	0.17 ± 0.40	0.22	0.85
All 2002	-0.07 ± 0.18	0.11	0.75

Table 6.3: Summary of the statistical results for correlations between Poynting flux and average neutral density deviation. Uncertainties are for a 95% confidence interval.

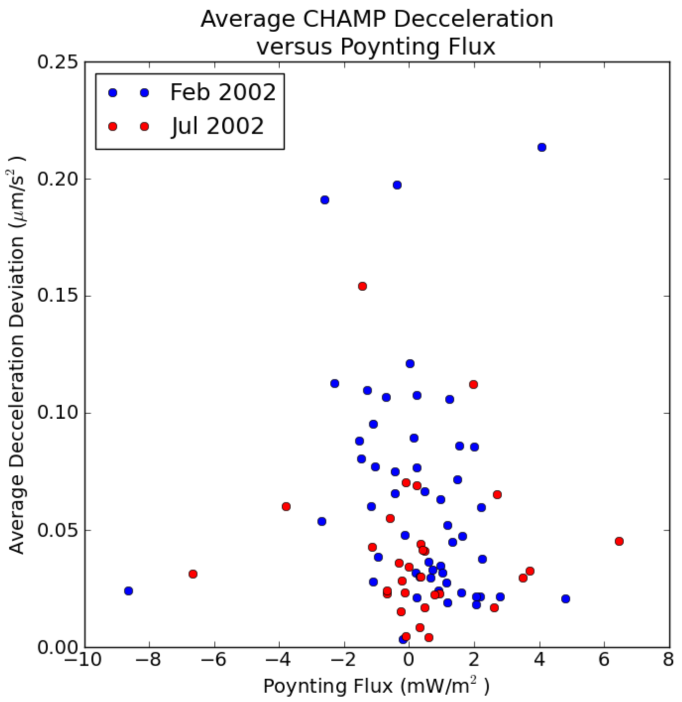
that electric field strengths were quite small during upwelling events [22]. In any case, even removing orbits with negative average Poynting flux yields no correlation for these data. (See Table 6.3.)

6.5.3 Electron Precipitation Density

Correlations between the neutral density and electron precipitation density showed little correlation. Correlations for each period individually were better, but the derived power law scalings varied widely. (See Fig. 6.14 and Table 6.4.)

6.5.4 Ion Outflow

Neither of the studied ionospheric energy inputs provide good correlation with neutral upwelling as observed by the decceleration of the CHAMP satellite. However, a good correlation does exist between the measured ion outflow by FAST with the CHAMP decceleration deviations. (See Fig. 6.15 and Table 6.5.) While more scatter is observed in this correlation than in those for the ion outflow data in the



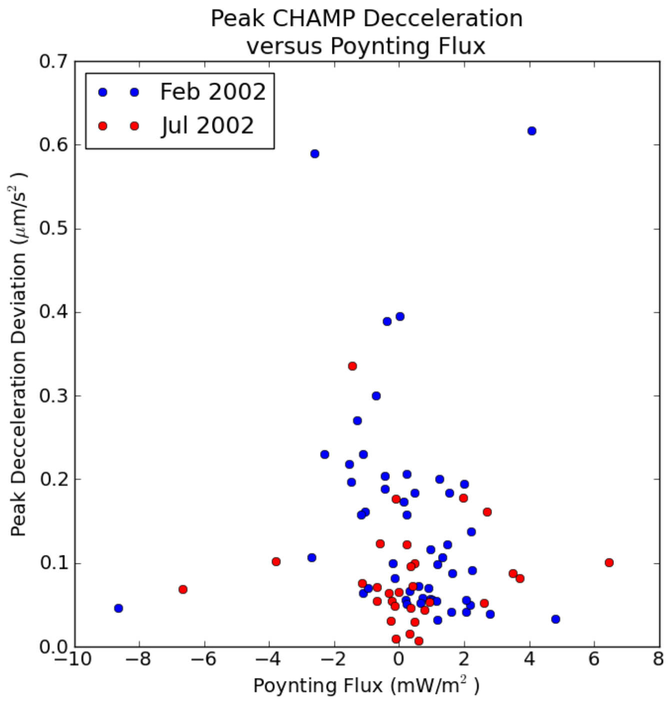
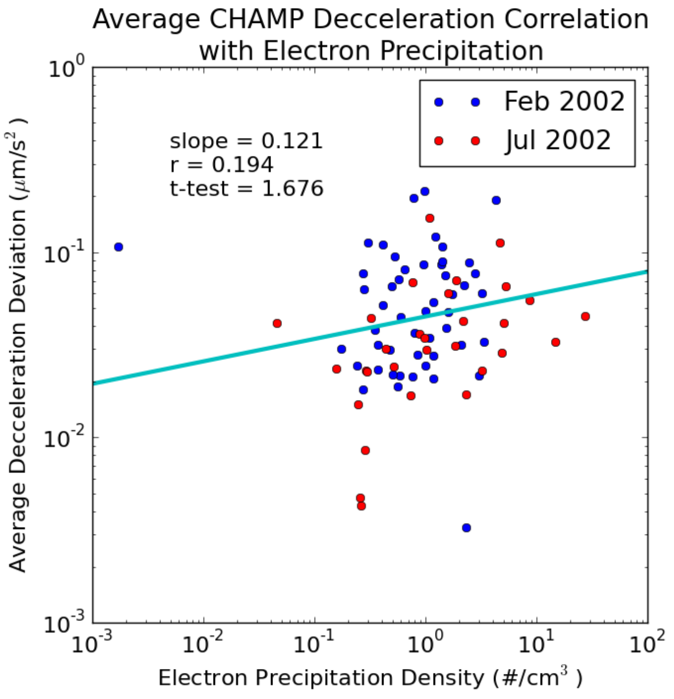


Figure 6.13: Plots of the average Poynting flux versus decceleration deviation (proportional to neutral density). Poynting flux shows no coorelation for both average deviation and peak deviation.



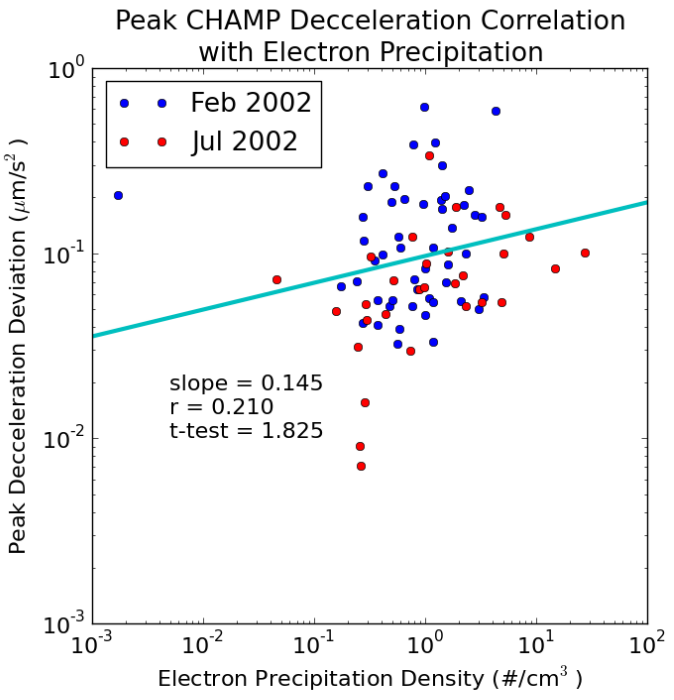


Figure 6.14: Plots of the average electron precipitation density versus decceleration deviation. Electron precipitation density shows a lack of correlation for both deviation measurements.

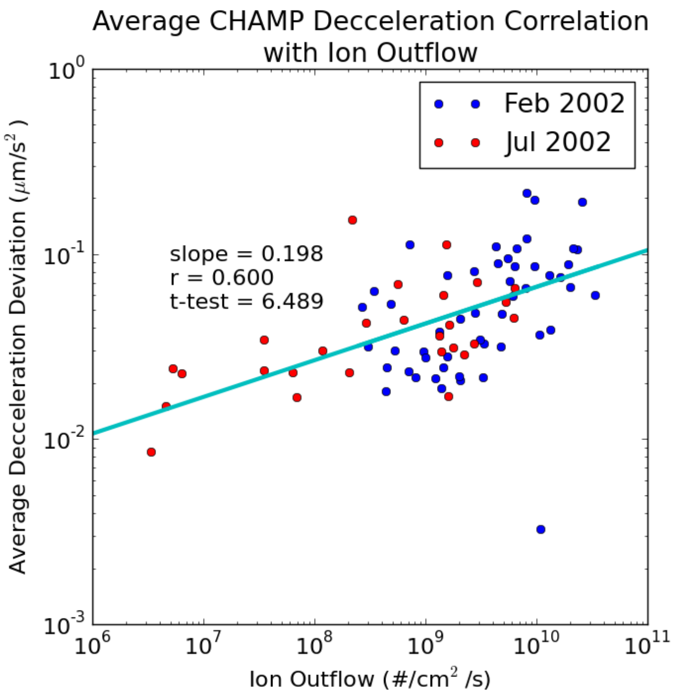
Data Set	*Slope*	*r-value*	*t-test*
Feb 2002	0.19 ± 0.24	0.22	1.55
Jul 2002	0.15 ± 0.17	0.36	1.87
All 2002	0.12 ± 0.14	0.19	1.68

Table 6.4: Summary of the statistical results for correlations between electron precipitation density and average neutral density deviation. Uncertainties are for a 95% confidence interval.

Data Set	*Slope*	*r-value*	*t-test*
Feb 2002	0.30 ± 0.12	0.58	4.96
Jul 2002	0.16 ± 0.09	0.59	3.61
All 2002	0.20 ± 0.06	0.60	6.49

Table 6.5: Summary of the statistical calculations for correlations between ion outflow and average neutral density deviation. Errors are for the 95% confidence interval.

previous sections, the data here are spread over nearly five orders of magnitude in ion outflow. A correlation of the ion outflow and neutral upwelling phenomena indicate some connection between the two, even though correlations with the energy sources that correlate with ion outflow do not exist for neutral upwelling.



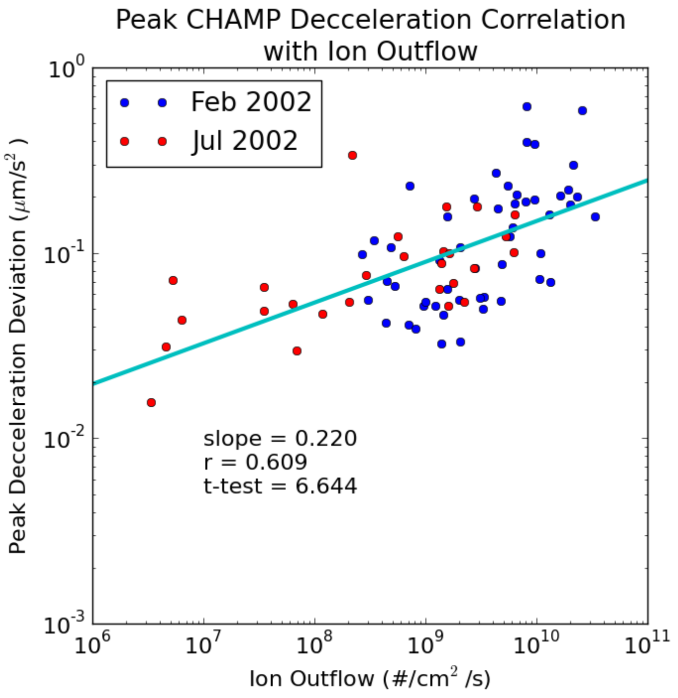


Figure 6.15: Plots of the average ion outflow versus decceleration deviation. Ion outflow has good correlation with both measurements.

6.6 Summary of Findings

This study finds agreement with the Strangeway results after using a larger data set centered on the September 1998 event. Both Poynting flux and electron precipitation density are found to correlate well with ion outflow flux in the dayside cusp. In addition, similar correlations are found for both ionospheric energy inputs with ion outflow fluxes in the two periods studied in 2002, though the Poynting flux data suffer from a faulty electric field probe on the FAST satellite. The noticeable difference between the February and July data is likely indicative of the different types of ionospheric conditions, as the July data is similar to the September 1998 data in ion outflow signatures. Finally, no strong correlations are observed between the energy inputs with neutral density in the dayside cusp, however a good correlation is found between ion outflow itself with the CHAMP decceleration.

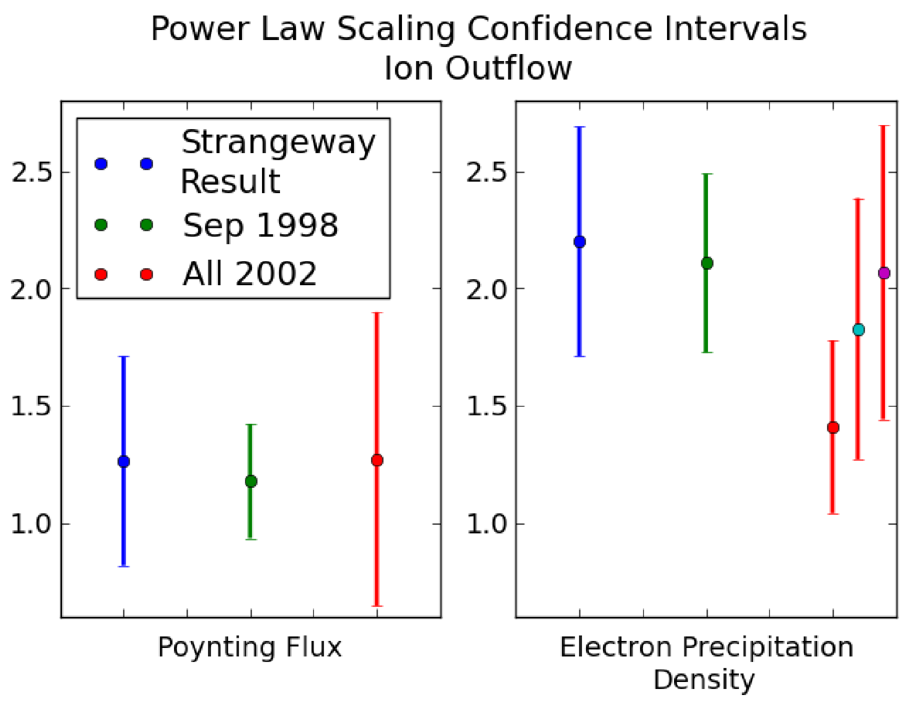


Figure 6.16: Summary of the statistical results for ion outflow, showing the power law scaling parameter with statistical uncertainty for a confidence interval of 95%. The overlapping confidence intervals indicate the consistent results, showing a power law scaling exponent of 1.2 for Poynting flux to ion outflow and 2.1 for electron precipitation to ion outflow (1.7 in 2002).

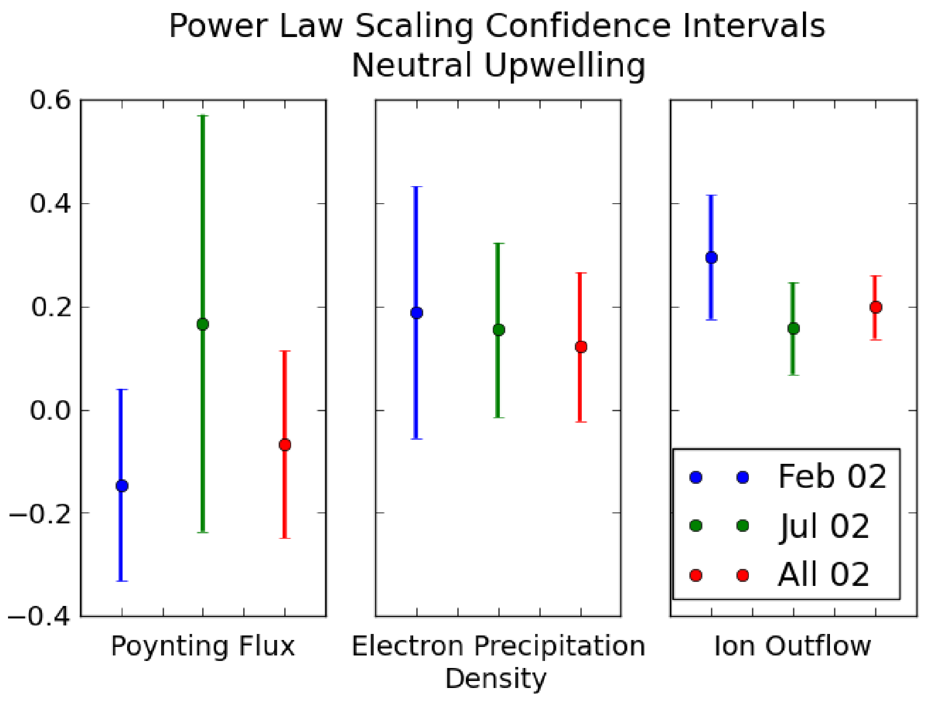


Figure 6.17: Summary of the statistical results for neutral upwelling, showing the power law scaling parameter with statistical uncertainty for a confidence interval of 95%. Poynting flux and electron precipitation scaling law exponents are near zero, and are considered uncorrelated. A scaling law exponent of 0.2 is found for the correlation between ion outflow and neutral density.

Chapter 7

Conclusion

Ultimately, the best way to determine the true causes and relationships between various ionospheric energy inputs and the ion outflow and neutral upwelling phenomena is to make *in situ* measurements of each property from the vantage point of a common spacecraft. Both sounding rocket and satellite platforms would be useful, since sounding rockets can target very specific locations and conditions for study, while satellites can aggregate a large quantity of data over a long period of time. Both would be very useful in understanding the neutral upwelling phenomenon, and it is hoped that in the future there will be a new RENU campaign.

7.1 The KIMS Instrument

The techonology to make these measurements is readily available to us today; we have presented one such instrument that would be very useful in the study of high energy ions in this context. This instrument (KIMS, as designated in this instance) is suited to a sounding rocket platform. KIMS provides unique benefits in its design, primarily as a means of assessing the energy distribution of the local ion plasma over short time scales because it is able to measure a range of energies simultaneously rather than requiring steps to sample the energy spectrum of the local plasma.

KIMS does have a few limitations, with its large mass and non-negligible external magnetic field. However, the external field is constant and small compared to the Earth's field, and careful measurements can provide necessary offsets for magnetometers. Particle instruments placed near the KIMS instrument may still be adversly affected, depending on the design, and so use of a KIMS type instrument necessitates careful planning and layout of a payload to minimize any adverse effects from the external magnetic fields.

Our experience in the assembly and testing of KIMS found that the magnetic circuit concept was very effective in practice. The assembly of the parts is straightforward, and the resulting magnetic fields are uniform and well-suited for effecting a separation of ions by their energy per charge. The calibration data showed very clearly that the overall instrument design worked as expected, and that the resulting response of the MCP detector matched the numerical model. If desired, a similar instrument design could be created for other energy ranges of ions, or even for electrons.

The MCP detector design and its accompanying electronics provide a lighter, less-expensive system than other particle detectors. The amplifier circuit design provides a fast, inexpensive pulse-counting system for such detectors and easily scales to any number of detector segments. The circuit design can easily be adjusted for other instruments used on sounding rocket payloads. The circuit design may also be of use for satellites with, at the added cost of further testing and use of radiation hardened components.

The electronics for the instrument were further simplified by using an FPGA for interfacing with the payload's telemetry unit. The GSE developed for testing and calibration made effective use of both FPGA and microcontroller design to interface the instrument with a computer.

7.2 Ion Outflow and Neutral Upwelling

Though the technology is available, the question still remains as to the value and feasibility of further missions to examine the neutral upwelling and ion outflow phenomena in the polar cusps. It is clear from the results of this study that the statistical relations derived by Strangeway *et al.* are consistent, and we expect there to be a power-law scaling for ion outflow from both Poynting flux and electron precipitation as energy sources (see Fig. 7.1). However, the amount of energy available in both of these sources is orders of magnitude more than is seen in the typical fluxes in ion outflow. The most likely place for the excess energy to go is the neutral gas in the ionosphere. It is reasonable to assume, then, that there is some electrodynamic contribution to the neutral upwelling phenomenon, and it may help to explain why this upwelling occurs.

A clear correlation is, in fact, found between the ion outflow and neutral upwelling phenomena. As the heating processes that lead to ion outflow are electrodynamic in nature, we expect there to be an electrodynamic connection to the heating of the neutral atmosphere, leading to the neutral upwelling phenomenon. Unfortunately, there is at best a poor correlation between the ionospheric energy sources considered in this work with the neutral density as measured by the CHAMP

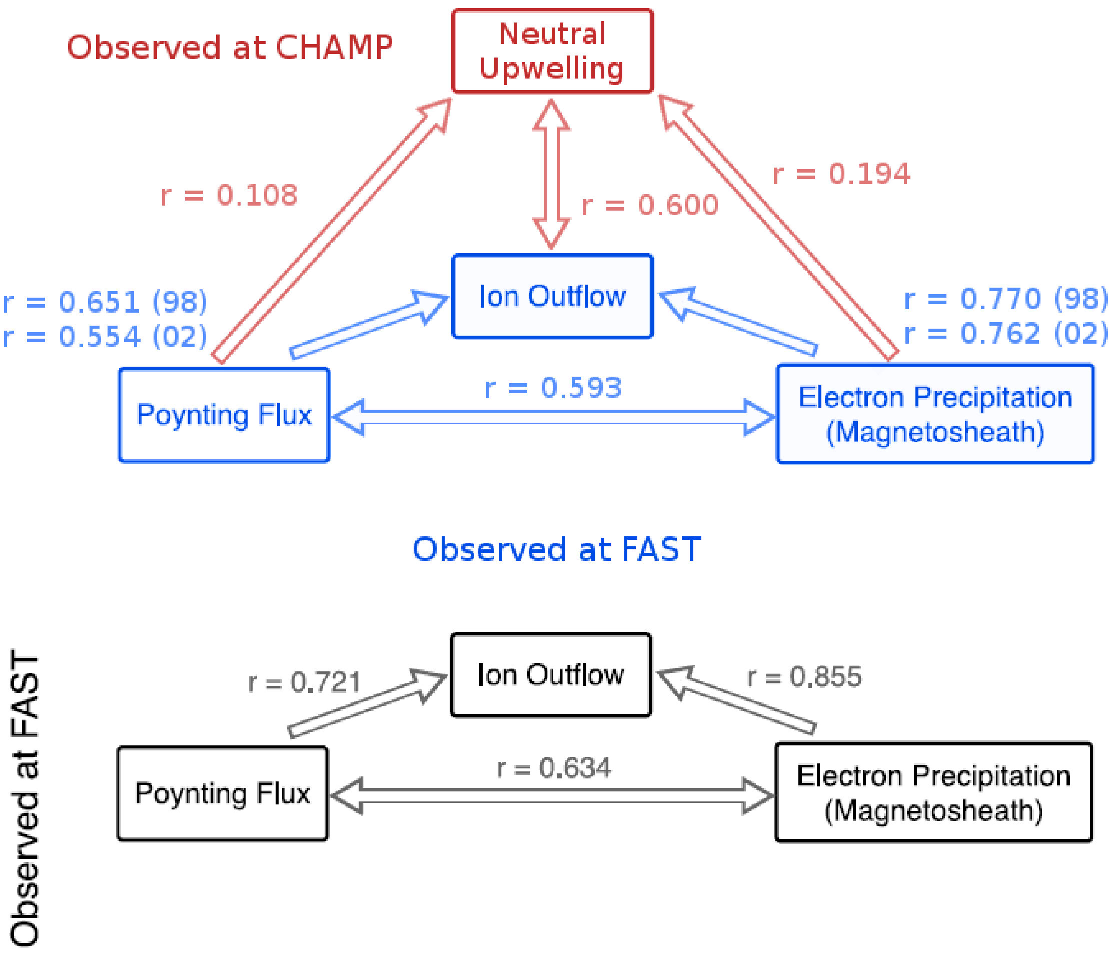


Figure 7.1: Figure similar to that in Strangeway *et al.*, 2005 [44], showing the correlations found in this study for the same ion outflow parameters (in blue) and for the neutral upwelling parameters (in red). In the case of neutral upwelling, only the ion outflow correlation was significant. The r-values from the ion outflow correlations compare well with those found in Strangeway (in black), as do the derived power law scaling values. Quoted values are for 2002 data unless otherwise specified.

satellite's accelerometer.

The most likely explanation of these results is that as energy is transferred to the ionospheric plasma from the fields and particles in the cusp, energy is also transferred to the neutral atmosphere as ion streams outflow from the ionosphere. In the case of the Poynting flux measurements, the low correlations observed may be due to compromised data or possibly a lack of Poynting flux during neutral upwelling events. The large spatial and temporal separation between the FAST and CHAMP satellites may be influential in the case of precipitating electrons. If field aligned currents at scales smaller than the spacecraft separation between the two satellites are driving the heating in the neutral atmosphere, this picture may be reasonable.

If, however, there is no correlation to be found between these energy sources and the neutral density, the correlation with ion outflow could indicate something entirely different, such as the outflowing ions themselves being the mechanism of transferring heat to the neutral gas. In this case, either the ions would be heated to well above escape velocity and lose the excess energy to the atmosphere, or the lost energy would be replenished through the same processes that drive the ion outflow in the first place.

What we can certainly conclude from these results is that there is good evidence for an electrodynamic connection between the two phenomena that justifies further study, preferably using *in situ* data from a common platform. A sounding rocket campaign such as RENU would be the most beneficial start in a further study searching for these connections.

7.3 Further Work

Were the KIMS instrument to be used in such a study, there are a few aspects where improvement could be made. Though the results of the calibration of this instrument show that we can be confident in the magnetic field strengths and uniformity we expected, it would be reassuring to have a calibrated measurement of the fields themselves. Ultimately the energy response of each anode segment is all that is necessary, so this measurement would be for further confidence in the overall instrument design. Use of a calibrated ion source would make it possible to obtain an absolute flux calibration for the instrument. A more comprehensive range of energies used in calibration would also be helpful, particularly at low energies. In addition, a more accurate scan in azimuth would lead to a more complete picture of the instrument response and would help in deciphering the actual data.

With little change to the instrument design itself, a very effective ion spectrograph could easily be made for another sounding rocket campaign. (An electron spectrograph could also be designed by using much lower magnetic fields in the magnetic circuit.) The instrument would fly in conjunction with other instruments, including other particle, neutral gas, and field detectors. Though there are many options for a high energy ion instrument, this design has excellent qualities that are beneficial to this type of study. A satellite platform may benefit from a lighter weight design, however.

Satellite measurements would also be very helpful in analyzing the two phenomena examined in this dissertation. Ideally the satellite would be able to provide all of the necessary data itself, but it is not unreasonable to use other satellites in conjunction to study these processes. The unfortunate loss of electric field data in the FAST satellite would suggest the need to use other satellites instead to further this study. Potentially useful satellites for this type of study may include the Defense Meteorological Satellite Program (DMSP) satellites. A new satellite, carrying a full suite of instrumentation in a polar orbit at an altitude between 300 and 500 km would be an ideal means of a long term study of the neutral upwelling phenomenon, but inclusion of these other satellites can extend our understanding.

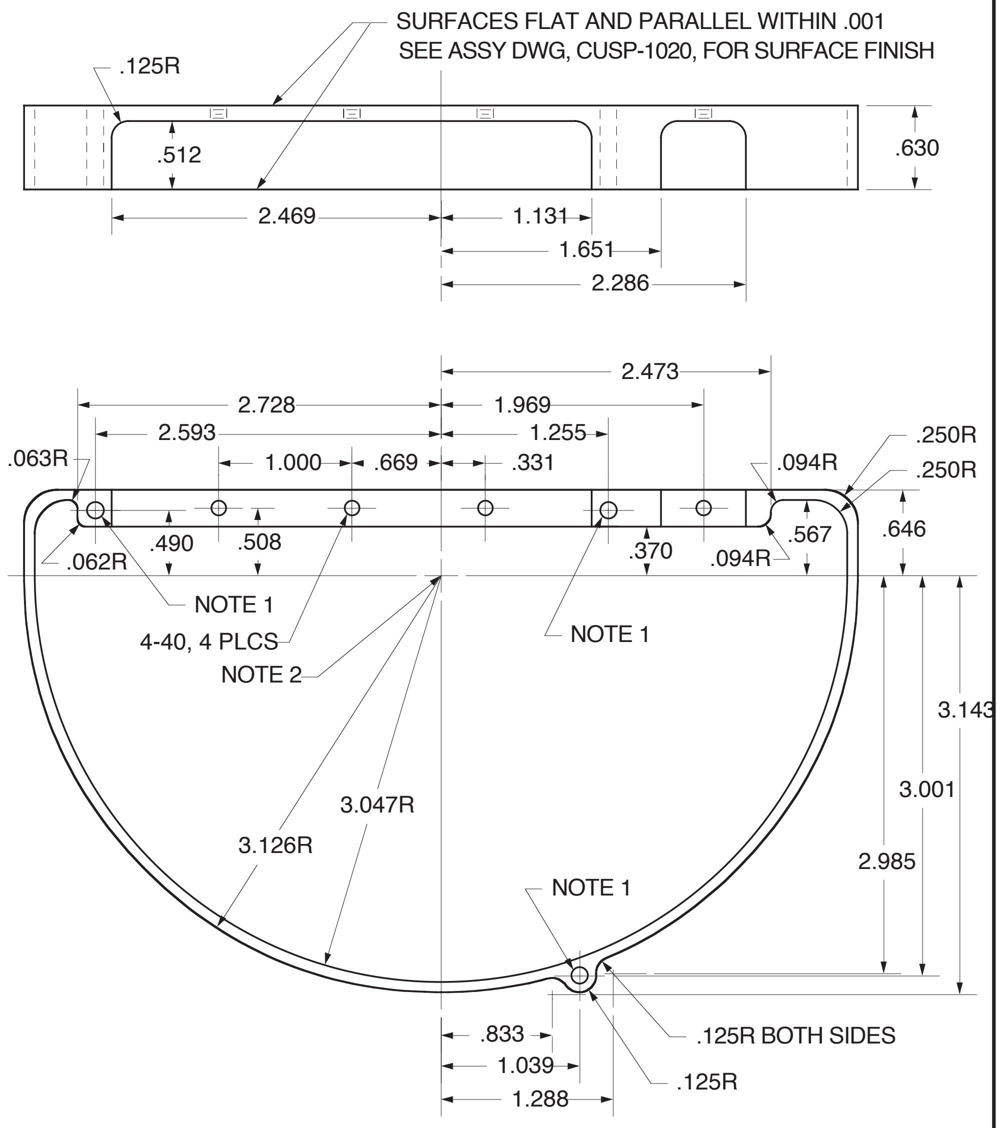
As we develop new methods of understanding how to correlate data sets from multiple spacecraft in environments as complex as the Earth's atmosphere, ionosphere, and magnetosphere, new insights and understandings will be discovered in relation to these phenomena as well as a number of others. The only way to completely untangle the complex interactions and changes that occur in these boundary regions is to examine both the broad and small scale properties simultaneously. While we continue to develop the tools and instruments necessary to undertake that task, there is much we can do with what we have. The statistical techniques used in this dissertation make it evident that there is a connection between the neutral upwelling and ion outflow phenomena. As we try to understand that connection, we will further our progress in being able to unravel these mysteries.

Appendix A

KIMS Hardware

A.1 Magnetic Circuit

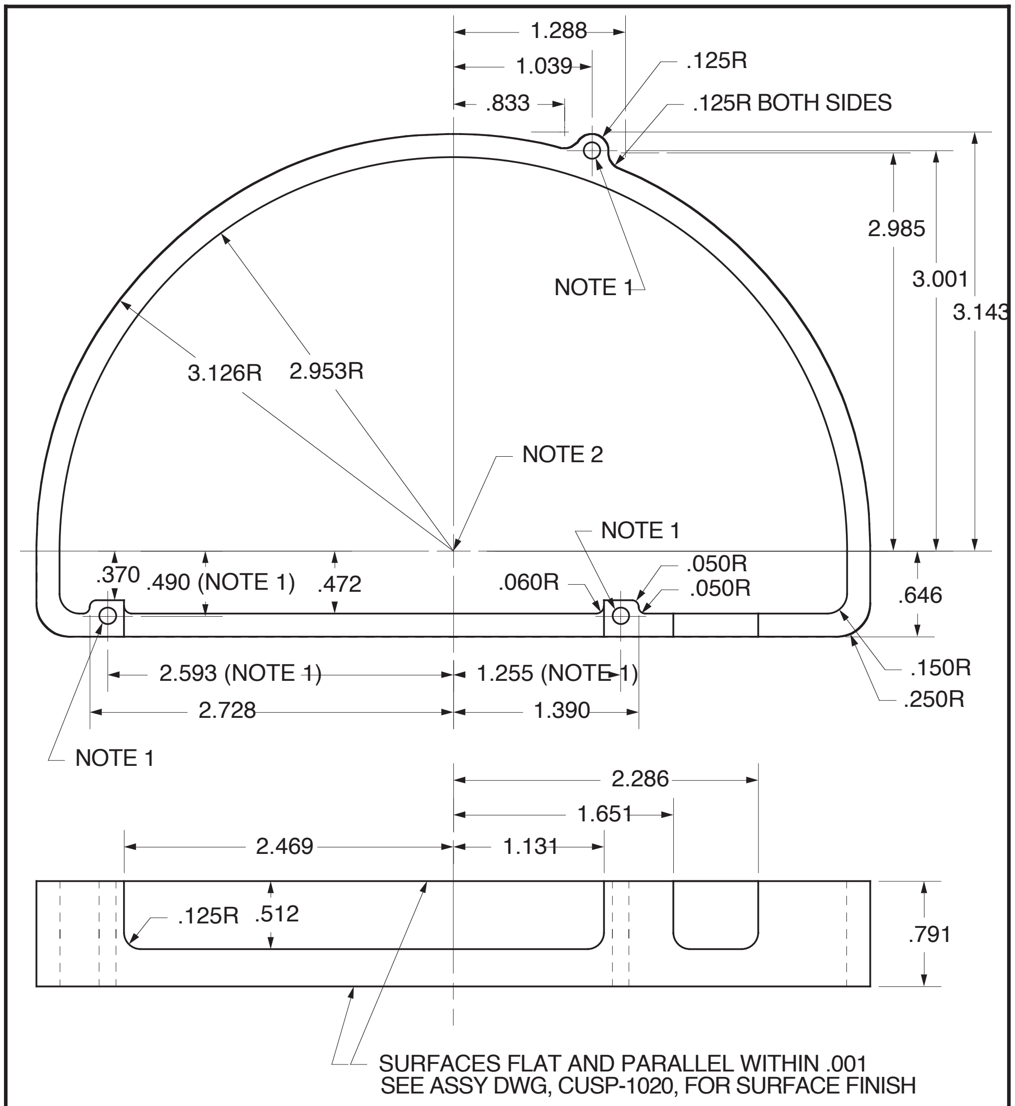
The following pages include the diagrams for the major components of the magnetic circuit, and the assembly checklist procedure for assembling the parts together. As the magnets are very strong, the checklist is recommended to ensure proper assembly the first time.



<u>NOTE 1</u>: THREE HOLES FOR 1/8D TYPE 302 STAINLESS STEEL ROLL PINS. THESE HOLES MUST ALIGN WITH HOLES IN PARTS CUSP-1016, CUSP-1017, CUSP-1018 AND CUSP-1019.

<u>NOTE 2</u>: PARTS CUSP-1015, CUSP-1016, CUSP-1017, CUSP-1018, AND CUSP-1019 ALIGN ON THIS CENTER.

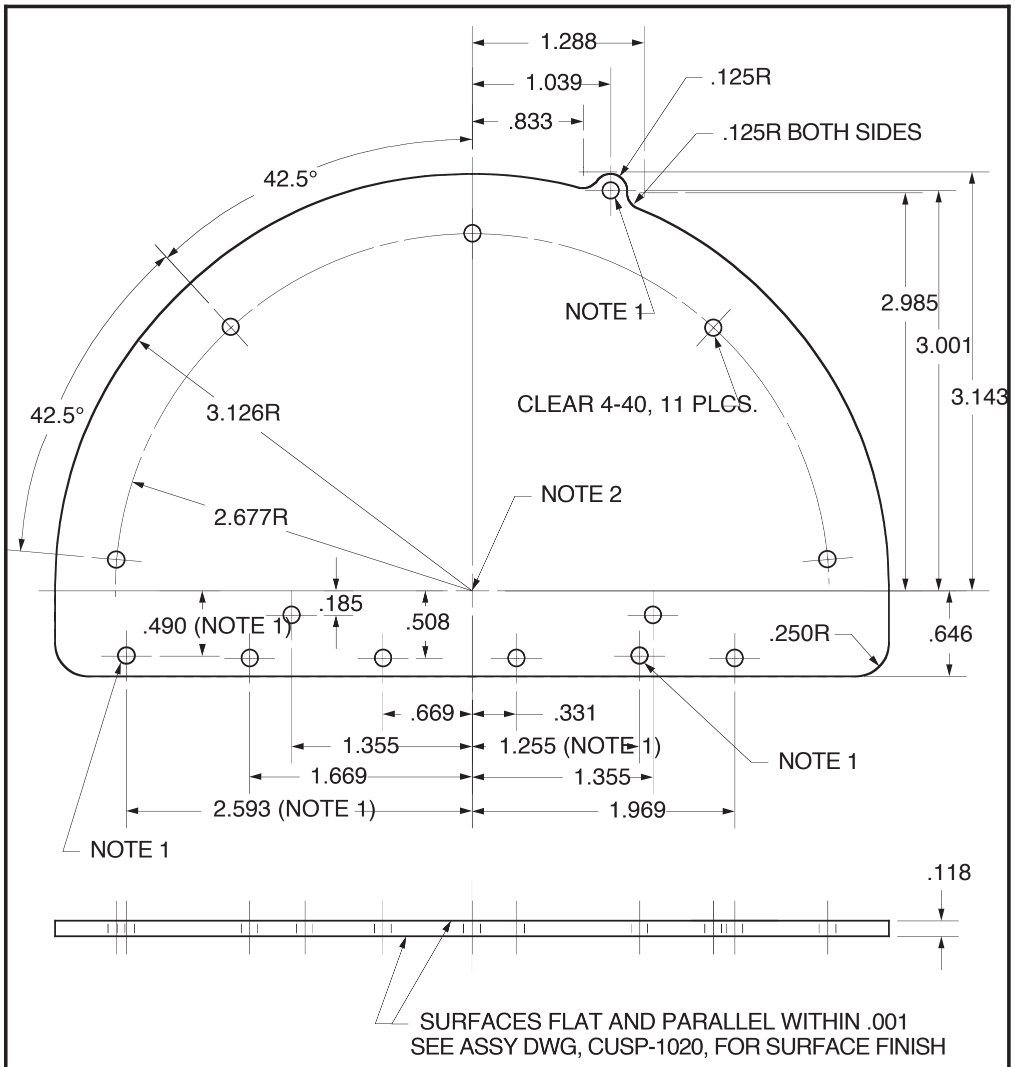
PART NO: CUSP-1015
TITLE: UPPER SHELL
NO. REQ'D: ONE
MAT'L: VANADIUM PERMENDUR
TOL: X.XXX = ±.001
DATE: 18 JAN 1997
U. MD. - J. H. MOORE - (301)-405-1



<u>NOTE 1</u>: THREE HOLES FOR 1/8D TYPE 302 STAINLESS STEEL ROLL PINS. THESE HOLES MUST ALIGN WITH HOLES IN PARTS CUSP-1015, CUSP-1017, CUSP-1018 AND CUSP-1019.

<u>NOTE 2</u>: PARTS CUSP-1015, CUSP-1016, CUSP-1017, CUSP-1018, AND CUSP-1019 ALIGN ON THIS CENTER.

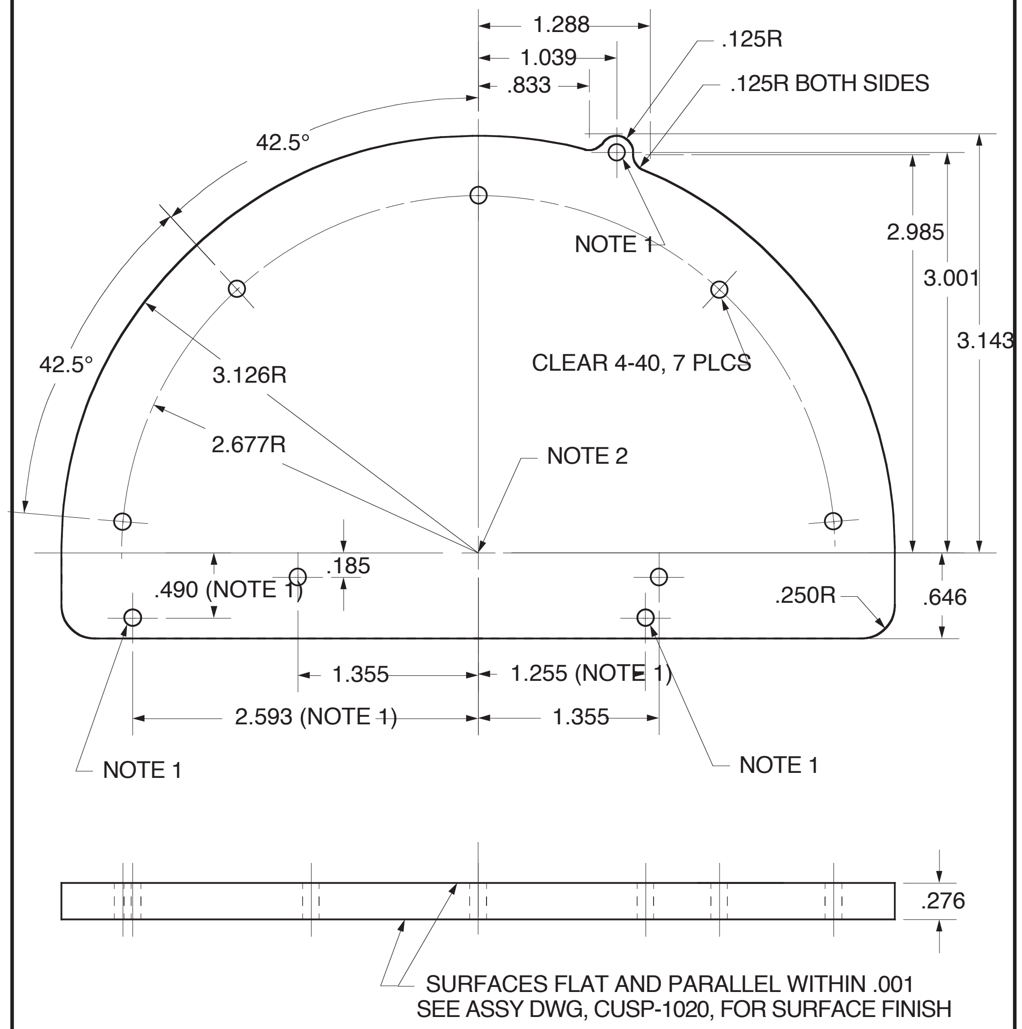
PART NO: CUSP-1016
TITLE: LOWER SHELL
NO. REQ'D: ONE MAT'L: VANADIUM PERMENDUR TOL: X.XXX = ±.001 DATE: 20 JAN 1997 U. MD. - J. H. MOORE - (301)-405-1



NOTE 1: THREE HOLES FOR 1/8D TYPE 302 STAINLESS STEEL ROLL PINS. THESE HOLES MUST ALIGN WITH HOLES IN PARTS CUSP-1015, CUSP-1016, CUSP-1018, AND CUSP-1019.

NOTE 2: PARTS CUSP-1015, CUSP-1016, CUSP-1017, CUSP-1018, AND CUSP-1019 ALIGN ON THIS CENTER.

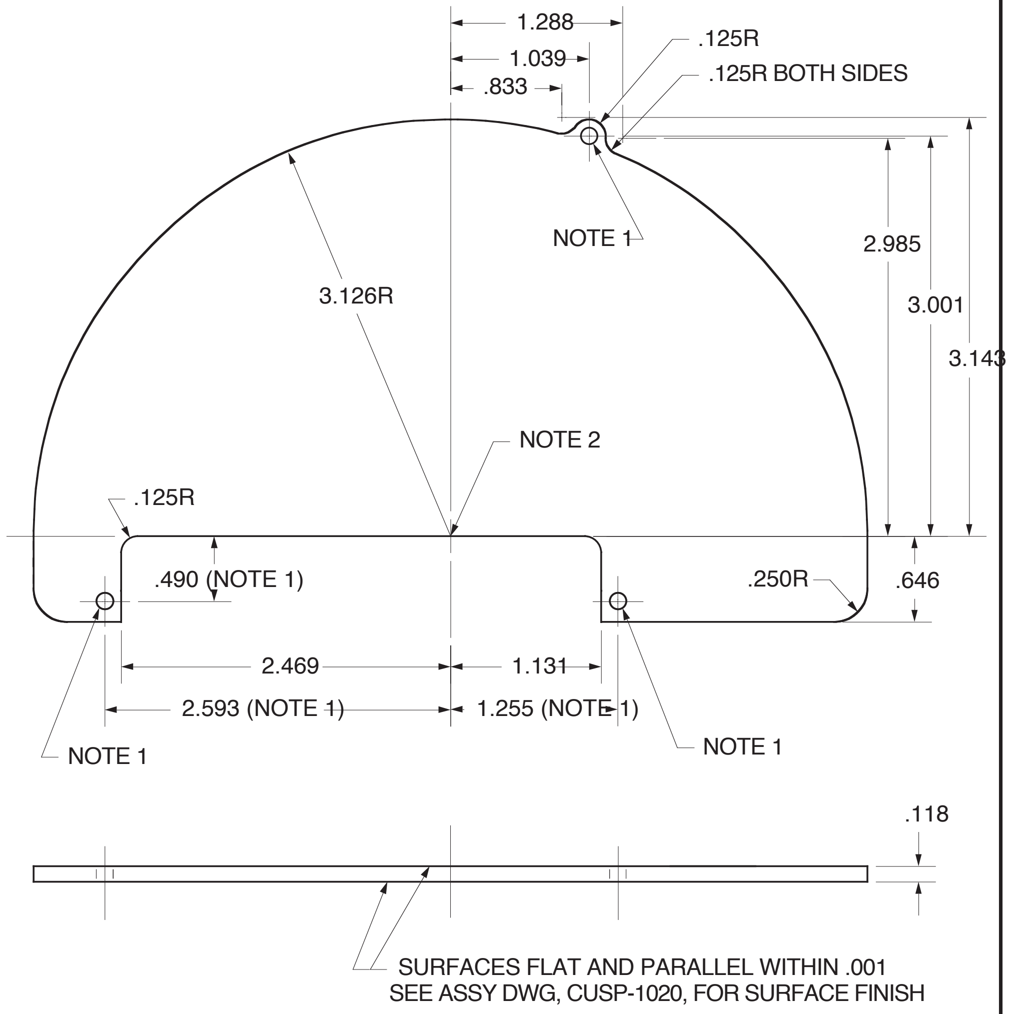
PART NO: CUSP-1017 (rev 1)
TITLE: UPPER COVER
NO. REQ'D: ONE
MAT'L: VANADIUM PERMENDUR
TOL: X.XXX = ±.001 ; X.X° = ± .02°
DATE: 20 JAN 1997
U. MD. - J. H. MOORE - (301)-405-1



NOTE 1: THREE HOLES FOR 1/8D TYPE 302 STAINLESS STEEL ROLL PINS. THESE HOLES MUST ALIGN WITH HOLES IN PARTS CUSP-1015, CUSP-1016, CUSP-1017 AND CUSP-1019.

NOTE 2: PARTS CUSP-1015, CUSP-1016, CUSP-1017, CUSP-1018, AND CUSP-1019 ALIGN ON THIS CENTER.

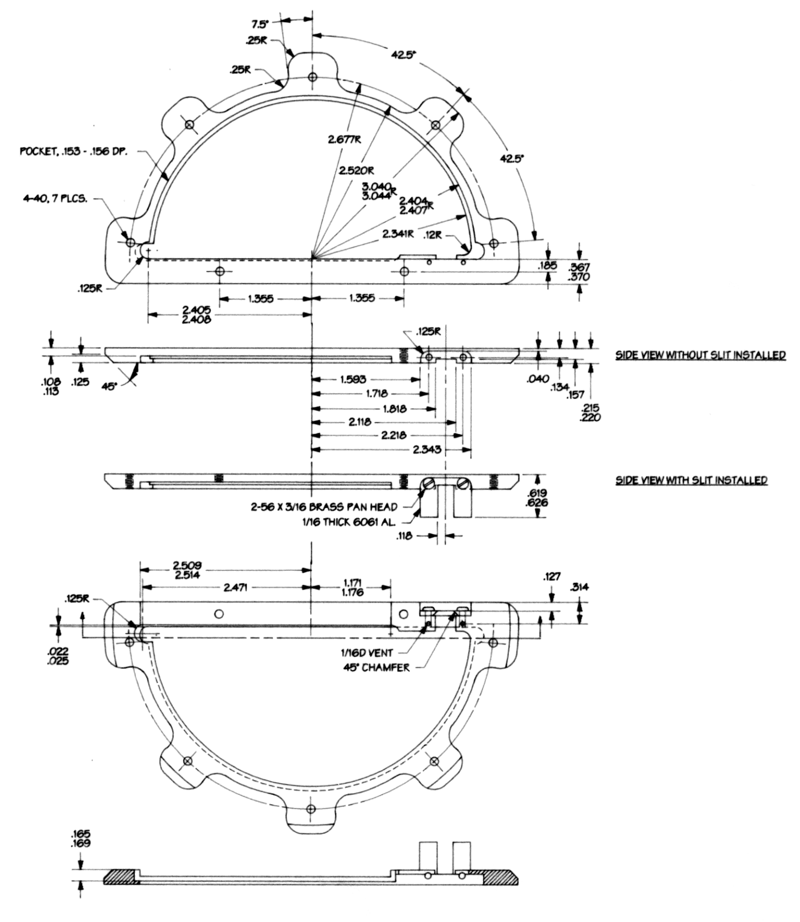
PART NO: CUSP-1018 (rev 1)
TITLE: LOWER COVER
NO. REQ'D: ONE MAT'L: VANADIUM PERMENDUR TOL: X.XXX = ±.001 ; X.X° = ±.02° DATE: 20 JAN 1997 U. MD. - J. H. MOORE - (301)-405-1



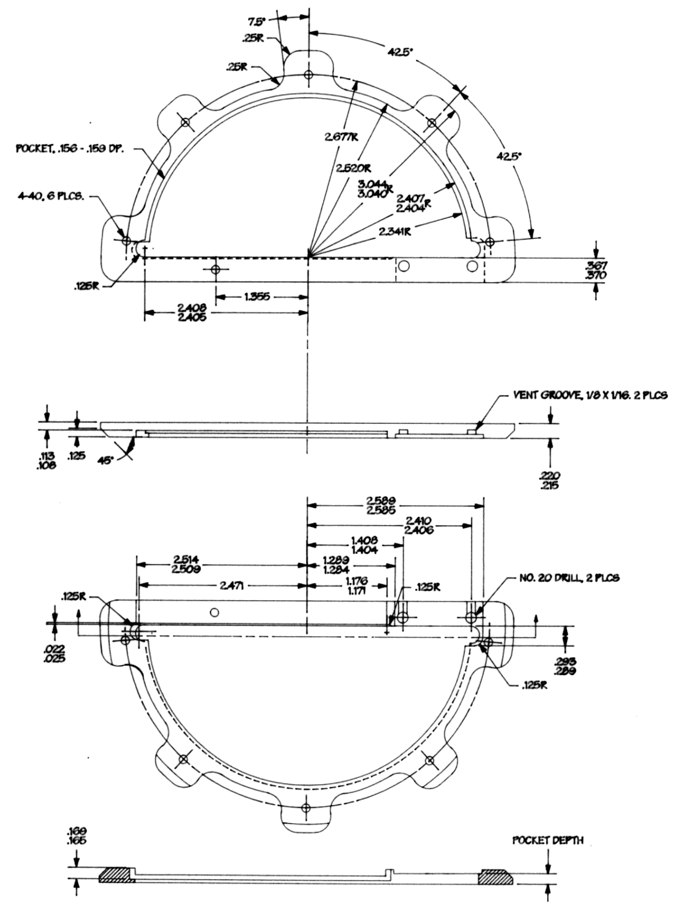
<u>NOTE 1</u>: THREE HOLES FOR 1/8D TYPE 302 STAINLESS STEEL ROLL PINS. THESE HOLES MUST ALIGN WITH HOLES IN PARTS CUSP-1015, CUSP-1016, CUSP-1017 AND CUSP-1018.

<u>NOTE 2</u>: PARTS CUSP-1015, CUSP-1016, CUSP-1017, CUSP-1018, AND CUSP-1019 ALIGN ON THIS CENTER.

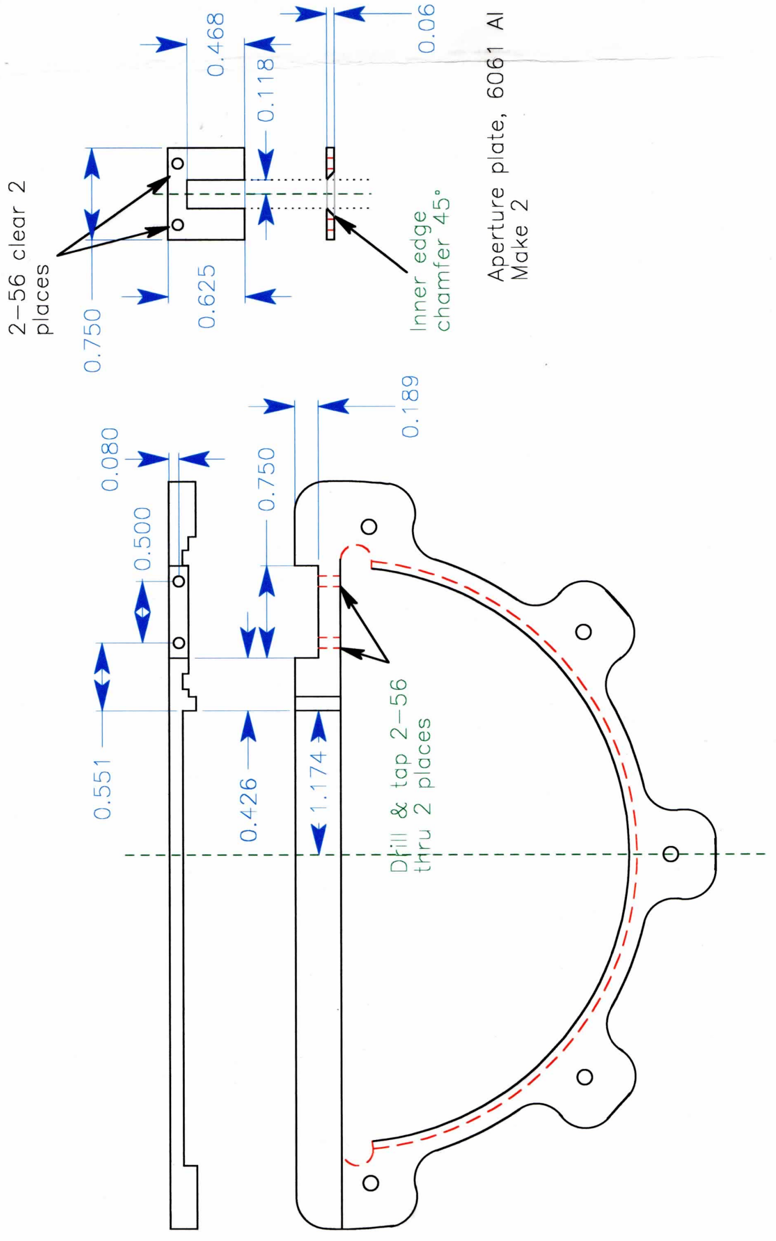
PART NO: CUSP-1019
TITLE: PARTITION
NO. REQ'D: ONE MAT'L: VANADIUM PERMENDUR TOL: X.XXX = ±.001 ; X.X° = ±.02° DATE: 20 JAN 1997 U. MD. - J. H. MOORE - (301)-405-1

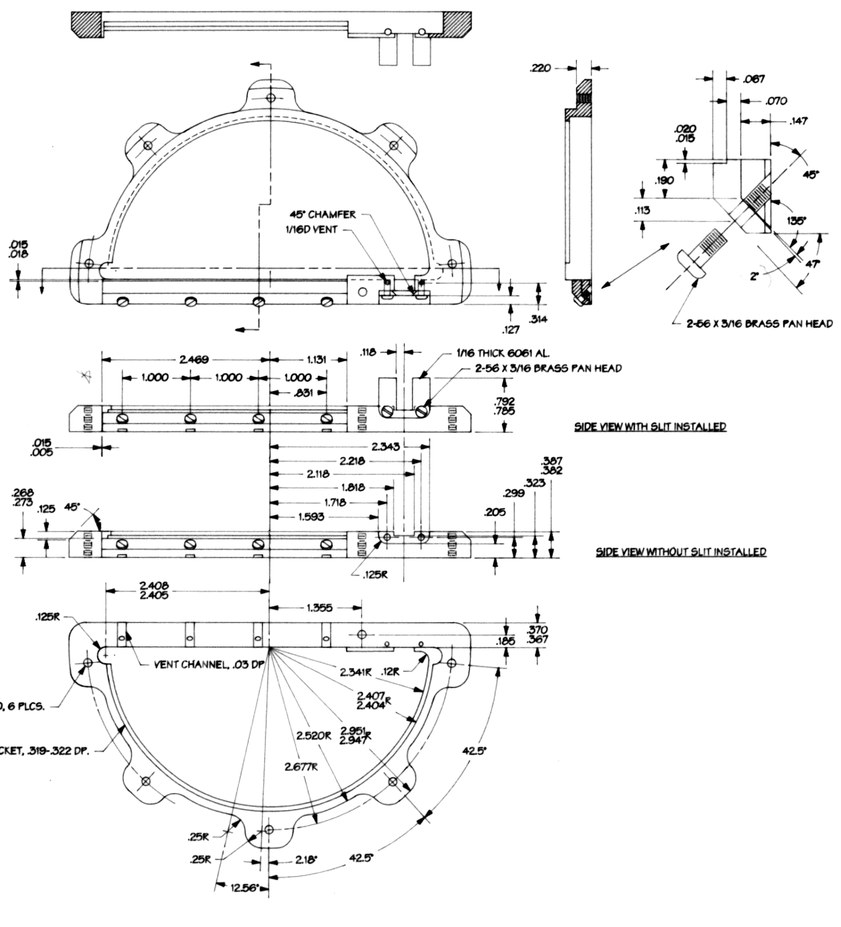


PART NO: CUSP-1034
TITLE: UPPER MAGNET CLAMP (WITH SLIT)
NO. REQ'D: ONE
MAT'L: 6061-T6 ALUMINUM ALLOY
TOL: X.XXX = ± .005; X.XX = ±.01; X/X = ±1/32
DATE: 9 MAY 1997
U. MD. - J. H. MOORE - (301)-405-1867

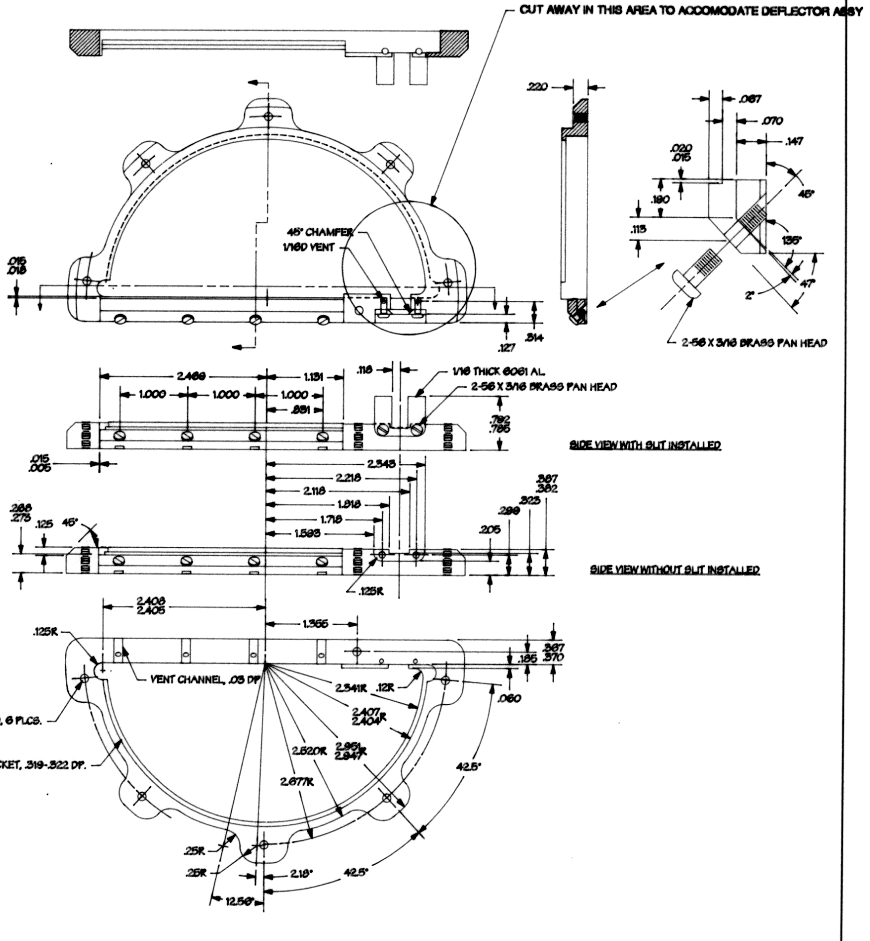


PART NO: CUSP-2034
TITLE: UPPER MAGNET CLAMP
NO. REQ'D: ONE MAT'L: 6061-T6 ALUMINUM ALLOY TOL: X.XXX = ± .005; X.XX = ±.01; X/X = ±1/32 DATE: 8 JAN 2001 U. MD. - J. H. MOORE - (301)-405-1867


0.06
0.468
0.118
2–56 clear 2 places
0.750
0.625
Inner edge chamfer 45°
Aperture plate, 6061 Al
Make 2
0.189
0.080
0.750
0.500
0.551
0.426
1.174
Drill & tap 2–56 thru 2 places

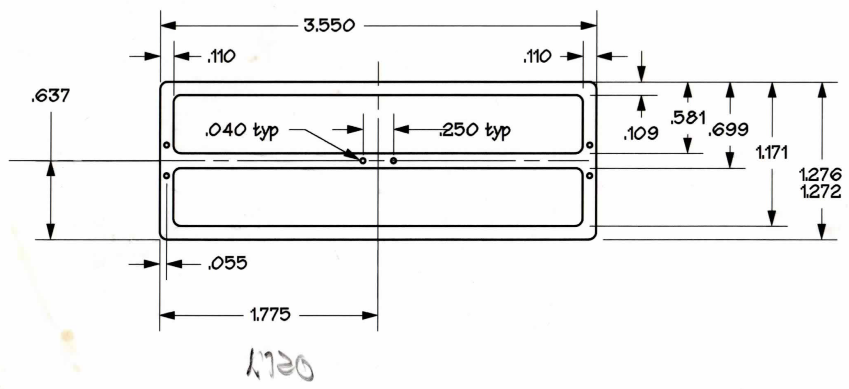


PART NO: CUSP-1036
TITLE: LOWER MAGNET CLAMP (WITH SLIT)
NO. REQ'D: ONE
MAT'L: 6061-T6 ALUMINUM ALLOY
TOL: X.XXX = ± .005; X.XX = ±.01; X/X = ±1/32
DATE: 16 MAY 1997
U. MD. - J. H. MOORE - (301)-405-1867



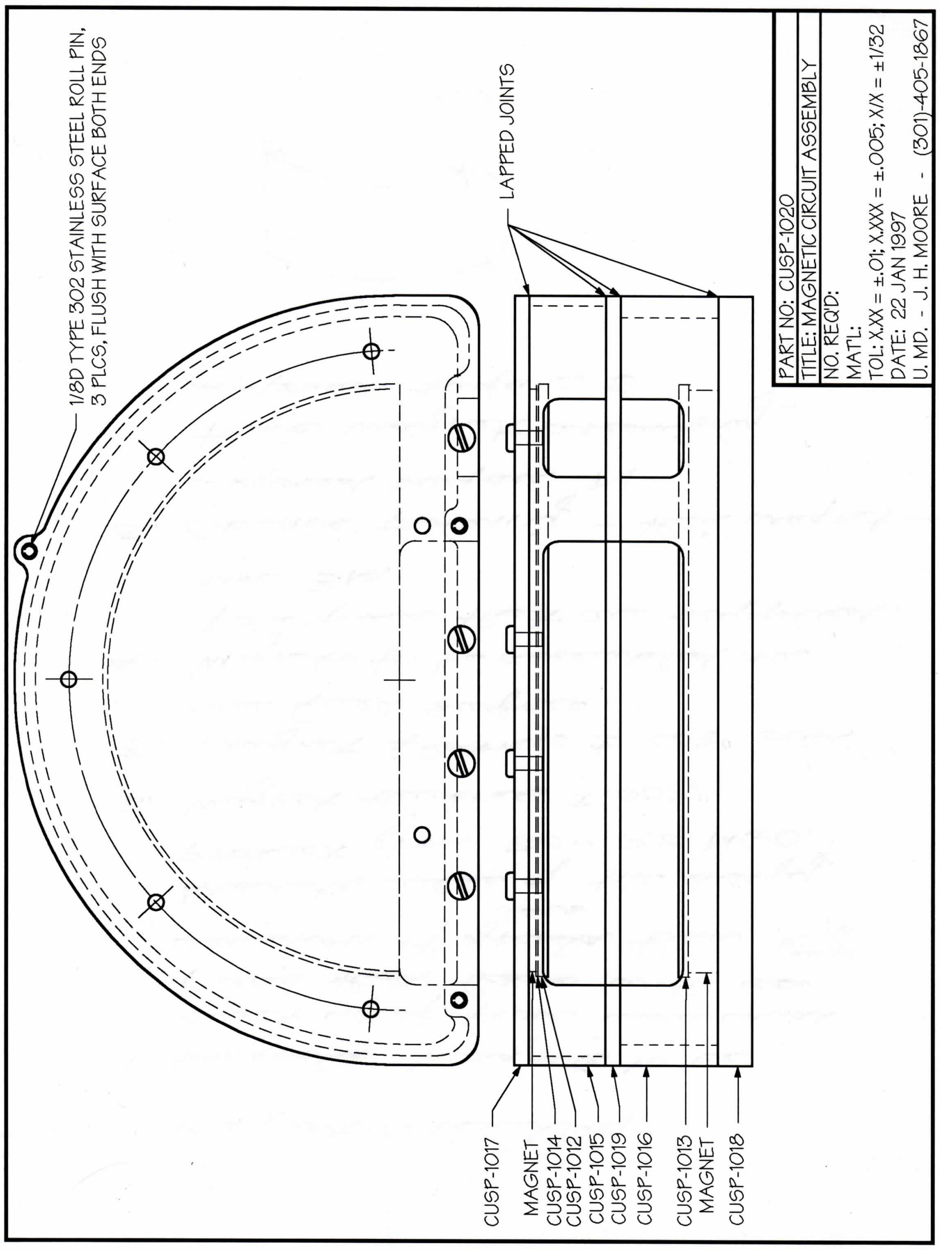
CUT AWAY IN THIS AREA TO ACCOMODATE DEFLECTOR ASSY
45° CHAMFER
1/16D VENT
2-56 X 3/16 BRASS PAN HEAD
1/16 THICK 6061 AL
2-56 X 3/16 BRASS PAN HEAD
SIDE VIEW WITH SLIT INSTALLED
SIDE VIEW WITHOUT SLIT INSTALLED
VENT CHANNEL, .03 DP
4-40, 6 PLCS.
POCKET, .319-.322 DP.

PART NO: CUSP-2036
TITLE: LOWER MAGNET CLAMP (WITH SLIT)
NO. REQ'D: 2 MAT'L: 6061-T6 ALUMINUM ALLOY TOL: X.XXX = ± .005; X.XX = ±.01; X/X = ±1/32 DATE: 10 JAN 2001 U. MD. - J. H. MOORE - (301)-405-1867



INSIDE AND OUTSIDE CORNERS: .06R

PART NO: CUSP-2047
TITLE: GROUND SCREEN FRAME
NO. REQ'D: 8
MAT'L: .010 ~~MOLYBDENUM~~ Phosphor Bronze
TOL: XXX = ±.01; XXXX = ±.005; X/X = ±1/32
DATE: 10 JULY 1997
U. MD. - J. H. MOORE - (301)-405-1867



PART NO: CUSP-1020
TITLE: MAGNETIC CIRCUIT ASSEMBLY
NO. REQ'D:
MAT'L:
TOL: X.XX = ±.01; X.XXX = ±.005; X/X = ±1/32
DATE: 22 JAN 1997
U. MD. - J. H. MOORE - (301)-405-1867

KIMS/Proton Magnetic Spectrograph
Magnetic Circuit Assembly
Rev. D 26 Jan 2010

1. Locate and clean:

- ◊ 1015 – Upper Shell
- ◊ 1016 – Lower Shell
- ◊ 1017 – Upper Cover
- ◊ 1018 – Lower Cover
- ◊ 1019 – Partition
- ◊ 1036/2036 – Lower Magnet Clamp
- ◊ Lower Aperture
- ◊ 1034/3034 – Modified Upper Magnet Clamp
- ◊ Upper Aperture
- ◊ Lower Magnet (Thick)
- ◊ Upper Magnet (Thin)
- ◊ 4-40 x 1” Slotted Screws (6)
- ◊ 4-40 x ½” Phillips Screws (8)
- ◊ 4-40 x 3/8” Phillips Screws (8)
- ◊ 4-40 x ¼” Phillips Screws (8)
- ◊ 2-56 x 3/16” Phillips Screws (6)
- ◊ 2-56 x ¼” Slotted Screws (8)
- ◊ 1/8” x 2” Roll Pins (5)
- ◊ 1/8” rods 6” long
- ◊ Jig Base
- ◊ Jig Nuts & Washers (4 ea.)
- ◊ Pads A, B, C, and D
- ◊ Spacers L and U
- ◊ Non-Magnetic Stainless Washers, 0.045” thickness (8)
- ◊ Non-Magnetic Slotted Screwdriver
- ◊ Non-Magnetic Phillips Screwdriver
- ◊ Non-Magnetic Needle Nose Pliers and/or Tweezers
- ◊ Non-Magnetic Mallet
- ◊ Chunk of plastic for setting pins
- ◊ 9V Battery
- ◊ Length of Insulated Wire

2. Lower Shell Assembly

◊ Put Lower Cover in jig, tab right.
◊ Place Pad A in jig (letter down).
◊ Thread and tighten 4 jig nuts on the jig bolts.
◊ Attach Lower Aperture to Lower Clamp, bevel in, with 2-56 x 3/16" screws and Loctite.
◊ Place Lower Magnet on the pad, pole face up.
◊ Place Lower Clamp over the Lower Magnet.
◊ Check that current running up the magnet (into the aperture) deflects left.
◊ Slide magnet onto the Lower Cover, carefully line up with holes 1-5.
◊ Thread holes 1 and 5 with ½" screws. (Screws will be upside down, and don't actually attach the clamp to the cover. This is only to keep the clamp from shifting on removal.)
◊ Remove Pad A.
◊ Remove Assembly.
◊ Thread holes 2 and 4 with 1" screws.
◊ Remove screws in 1 and 5.
◊ Replace Lower Assembly in jig, tab right.
◊ Attach Spacer L and thread holes 1, 3, and 5 with 1" screws.
◊ Slide Lower Shell onto assembly, aperture to the right.
◊ Remove screws in 1, 3, and 5.
◊ Remove Spacer L.
◊ Remove Assembly.
◊ Thread holes 1, 3, 5, and 6 with ½" screws.
◊ Remove screws in 2 and 4, replace with ½" screws.
◊ Carefully set Lower Assembly to the side, away from any magnetic materials.

3. Upper Shell Assembly

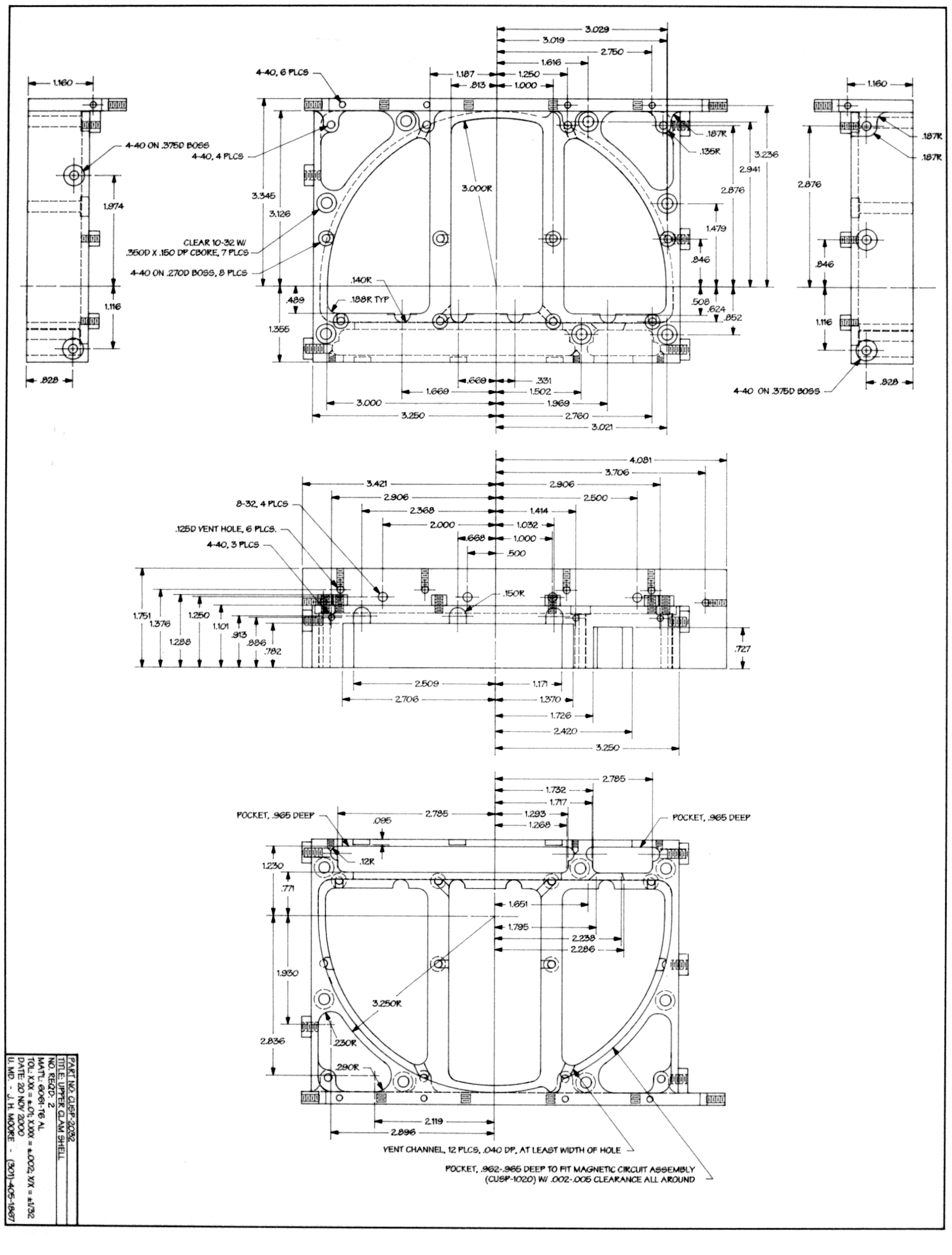
- ◊ Place Upper Cover in jig, tab left.
- ◊ Place Pad D in jig (letter down).
- ◊ Thread and tighten 4 jig nuts on the jig bolts.
- ◊ Attach Upper Aperture to Upper Clamp, bevel in, with 2-56 x ¼” screws and Loctite.
- ◊ Place Upper Magnet on the pad, pole face up.
- ◊ Place Upper Clamp over the Upper Magnet.
- ◊ Check that the current running up the magnet (into the aperture) deflects right.
- ◊ Slide magnet onto the Upper Cover, carefully line up with holes 1-5.
- ◊ Thread holes 1 and 5 (screws upside down).
- ◊ Remove Pad D.
- ◊ Remove Assembly.
- ◊ Thread screws 2 and 4 with 1” long screws.
- ◊ Remove screws in 1 and 5.
- ◊ Thread hole 6 with 1” long screws (upside down), flush to top of cover.
- ◊ Attach Spacer U and thread holes 1, 3, and 5 with 1” screws.
- ◊ Slide Upper Shell onto assembly, aperture to the left.
- ◊ Remove screws in 1, 3, and 5.
- ◊ Remove Spacer U
- ◊ Remove screw in 6.
- ◊ Remove assembly.
- ◊ Thread holes 1, 3, and 5 with 3/8” screws.
- ◊ Remove screws in 2 and 4.
- ◊ Thread holes 2 and 4 with 3/8” screws.

4. Circuit Assembly

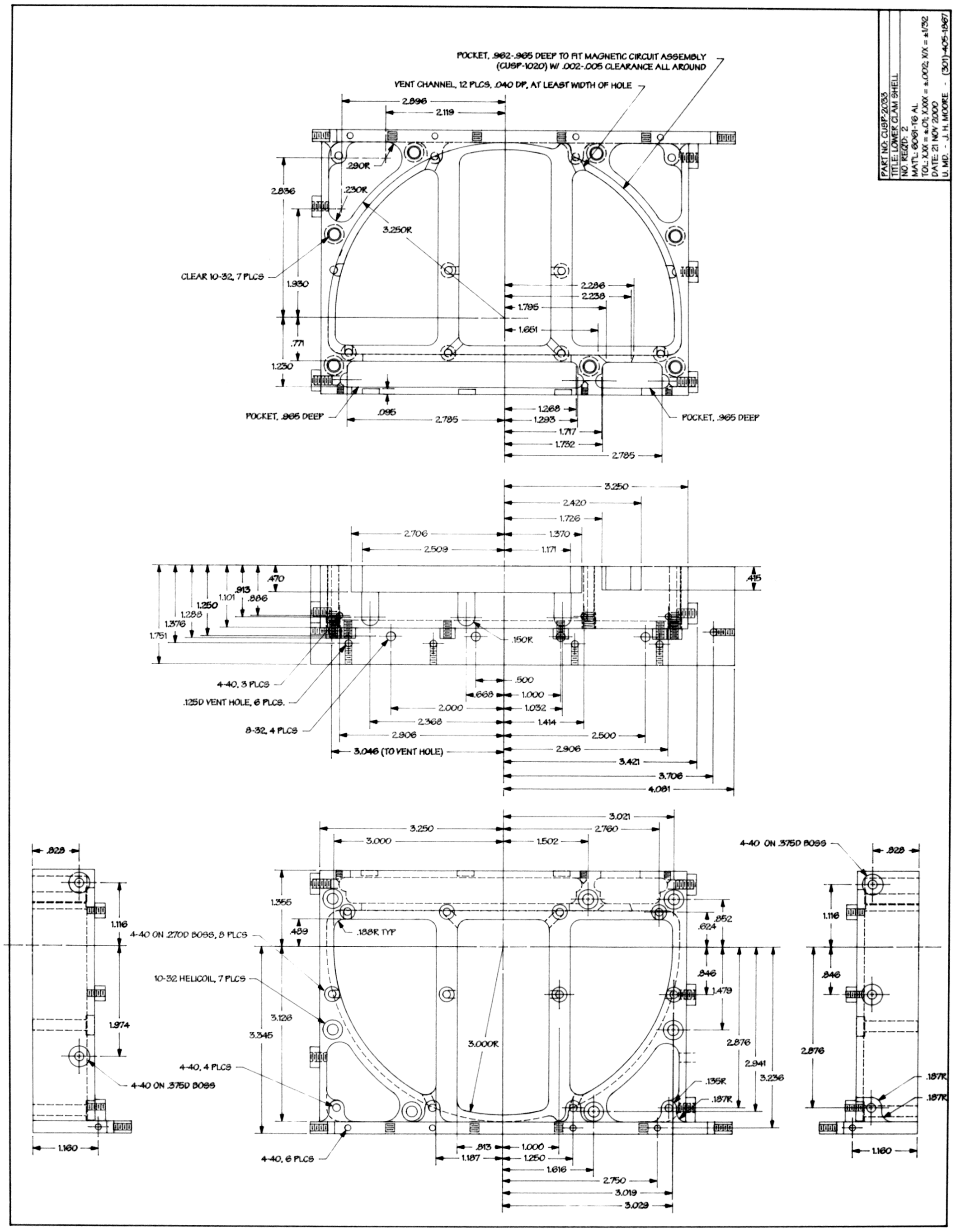
- ◊ Place Partition in jig, tab right.
- ◊ Place Pad D in jig (letter down).
- ◊ Thread and tighten 4 jig nuts on the jig bolts.
- ◊ Slide Upper Assembly onto Partition.
- ◊ Remove Pad D
- ◊ Remove Upper Assembly with Partition.
- ◊ Place 2 Washers ea. on jig bolts.
- ◊ Replace Upper Assembly with tab left; the partition will be on top.
- ◊ Place Pads A, B, C, and D in jig.
- ◊ Thread and tighten 4 jig nuts on the jig bolts.
- ◊ Slide Lower Assembly onto the partition.
- ◊ Insert pins 1, 2, and 3.
- ◊ Remove Pads A, B, C, and D.
- ◊ Remove Finished Circuit and Inspect.
- ◊ Thread Upper holes 5, a, b, c, and d with ¼" screws.
- ◊ Wrap carefully for storage to prevent contamination of the magnetic poles.

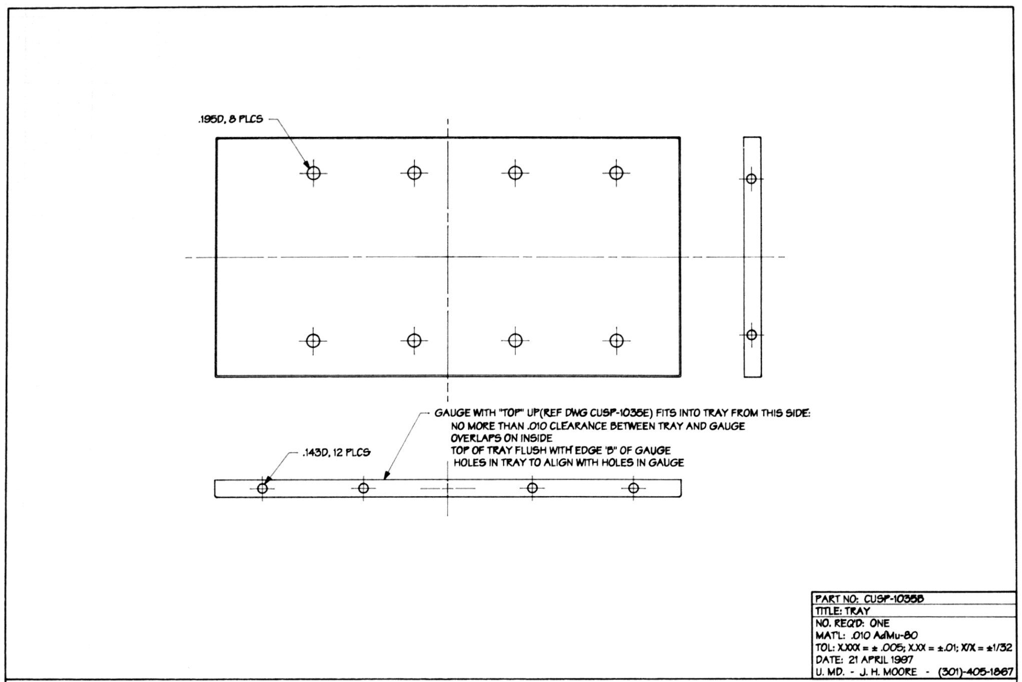
A.2 Clamshell and Cover

The following pages include the diagrams for the clamshell and cover for the KIMS instrument.

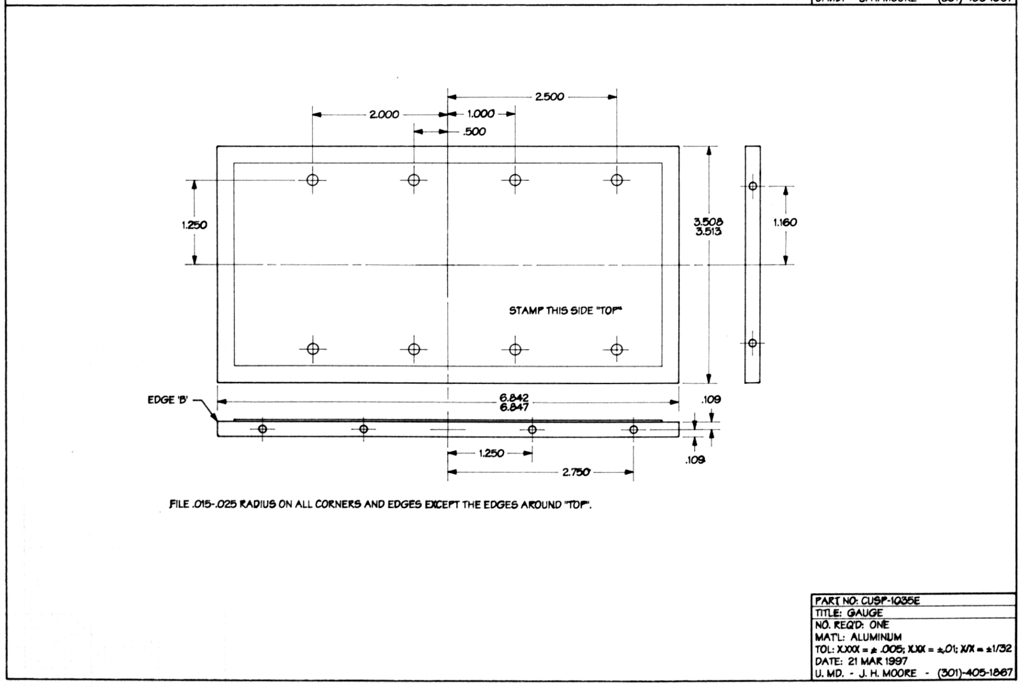

4-40, 6 PLCS
4-40 ON .375D BOSS
4-40, 4 PLCS
CLEAR 10-32 W/ .350D X .150 DP CBORE, 7 PLCS
4-40 ON .270D BOSS, 8 PLCS
3.000R
.140R
.188R TYP
.187R
.135R
8-32, 4 PLCS
.125D VENT HOLE, 6 PLCS.
4-40, 3 PLCS
.150R
POCKET, .965 DEEP
.12R
3.250R
.230R
.290R
VENT CHANNEL, 12 PLCS, .040 DP, AT LEAST WIDTH OF HOLE
POCKET, .962-.965 DEEP TO FIT MAGNETIC CIRCUIT ASSEMBLY (CUSP-1020) W/ .002-.005 CLEARANCE ALL AROUND

PART NO. CUSP-2032
TITLE: UPPER CLAM SHELL
NO. REQD: 2
MAT'L: 6061-T6 AL
TOL: X.XX = ±.01; X.XXX = ±.002; X/X = ±1/32
DATE: 20 NOV 2000
U. MD. - J. H. MOORE - (301)-405-1867


PART NO: CUSP-2033
TITLE: LOWER CLAM SHELL
NO. REQD: 2
MATL: 6061-T6 AL
TOL: X.XX = ±.01; X.XXX = ±.002; X/X = ±1/32
DATE: 21 NOV 2000
U. MD. - J. H. MOORE - (301)-405-1867
POCKET, .962-.965 DEEP TO FIT MAGNETIC CIRCUIT ASSEMBLY (CUSP-1020) W/ .002-.005 CLEARANCE ALL AROUND
VENT CHANNEL, 12 PLCS, .040 DP, AT LEAST WIDTH OF HOLE
CLEAR 10-32, 7 PLCS
POCKET, .965 DEEP
POCKET, .965 DEEP
4-40, 3 PLCS
.125D VENT HOLE, 6 PLCS.
8-32, 4 PLCS
3.046 (TO VENT HOLE)
4-40 ON .270D BOSS, 8 PLCS
10-32 HELICOIL, 7 PLCS
4-40, 4 PLCS
4-40, 6 PLCS
4-40 ON .375D BOSS
4-40 ON .375D BOSS
.188R TYP
3.000R
3.250R
.290R
.230R
.150R
.135R
.187R

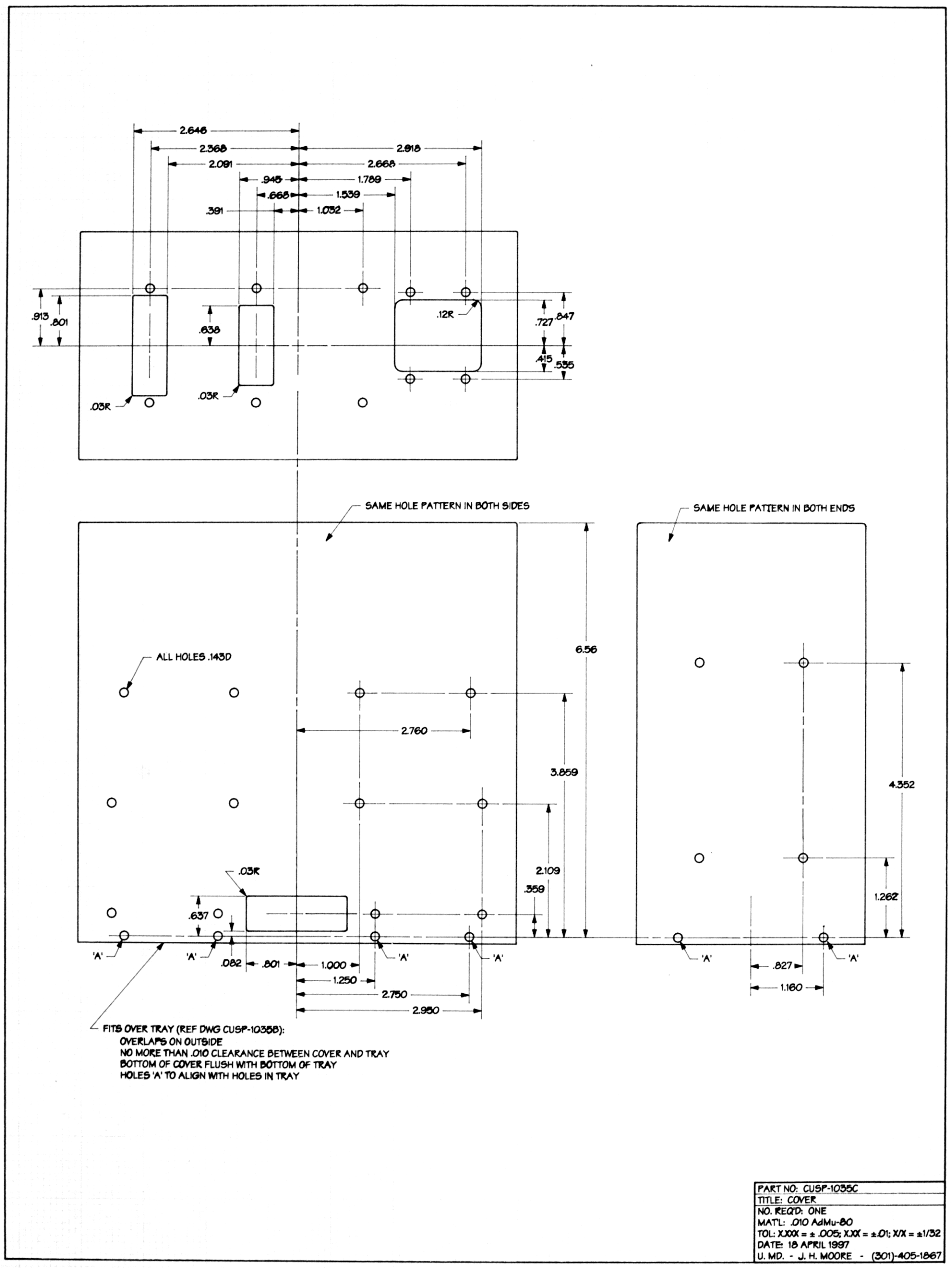

.195D, 8 PLCS
GAUGE WITH "TOP" UP(REF DWG CUSP-1035E) FITS INTO TRAY FROM THIS SIDE:
NO MORE THAN .010 CLEARANCE BETWEEN TRAY AND GAUGE
OVERLAPS ON INSIDE
TOP OF TRAY FLUSH WITH EDGE 'B" OF GAUGE
HOLES IN TRAY TO ALIGN WITH HOLES IN GAUGE
.143D, 12 PLCS

PART NO: CUSP-1035B
TITLE: TRAY
NO. REQ'D: ONE
MAT'L: .010 AdMu-80
TOL: X.XXX = ± .005; X.XX = ±.01; X/X = ±1/32
DATE: 21 APRIL 1997
U. MD. - J. H. MOORE - (301)-405-1867

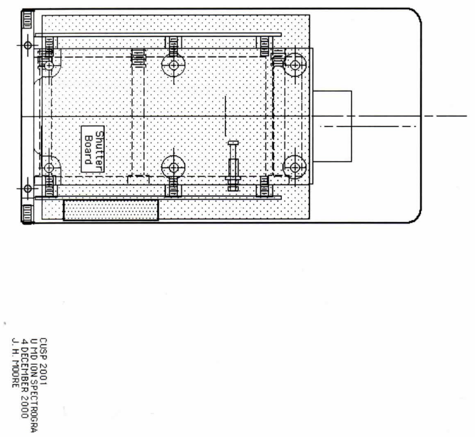


FILE .015-.025 RADIUS ON ALL CORNERS AND EDGES EXCEPT THE EDGES AROUND "TOP".

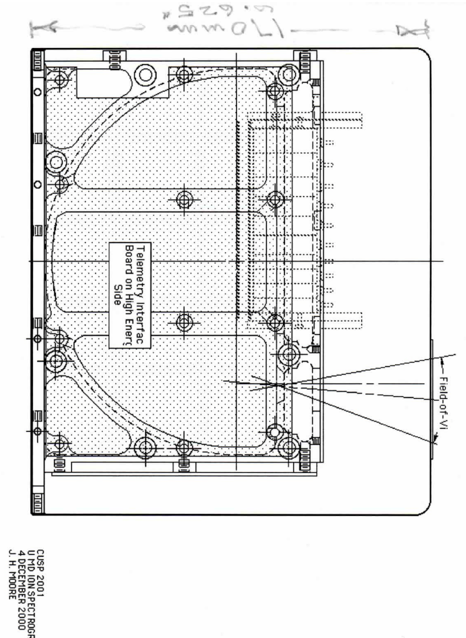
PART NO: CUSP-1035E
TITLE: GAUGE
NO. REQ'D: ONE
MAT'L: ALUMINUM
TOL: X.XXX = ± .005; X.XX = ±.01; X/X = ±1/32
DATE: 21 MAR 1997
U. MD. - J. H. MOORE - (301)-405-1867


2.646
2.368
2.091
.946
.668
.391
2.918
2.668
1.789
1.539
1.032
.913
.801
.638
.12R
.727
.847
.415
.535
.03R
.03R
SAME HOLE PATTERN IN BOTH SIDES
SAME HOLE PATTERN IN BOTH ENDS
ALL HOLES .143D
6.56
2.760
3.859
2.109
.359
.03R
.637
'A'
'A'
'A'
'A'
.082
.801
1.000
1.250
2.750
2.950
4.352
1.262
'A'
'A'
.827
1.160
FITS OVER TRAY (REF DWG CUSP-1035B):
OVERLAPS ON OUTSIDE
NO MORE THAN .010 CLEARANCE BETWEEN COVER AND TRAY
BOTTOM OF COVER FLUSH WITH BOTTOM OF TRAY
HOLES 'A' TO ALIGN WITH HOLES IN TRAY

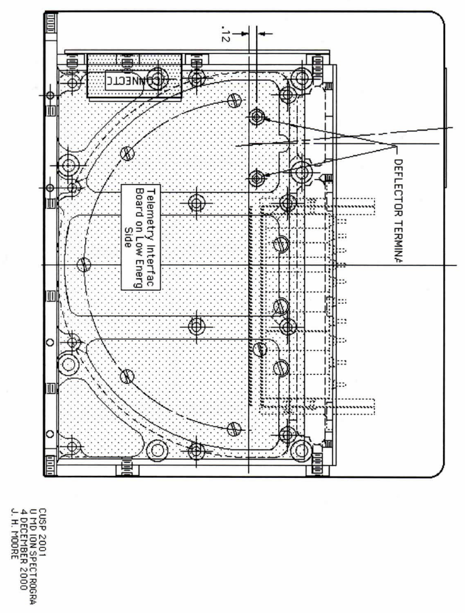
PART NO: CUSP-1035C
TITLE: COVER
NO. REQ'D: ONE
MAT'L: .010 AdMu-80
TOL: X.XXX = ± .005; X.XX = ±.01; X/X = ±1/32
DATE: 18 APRIL 1997
U. MD. - J. H. MOORE - (301)-405-1867



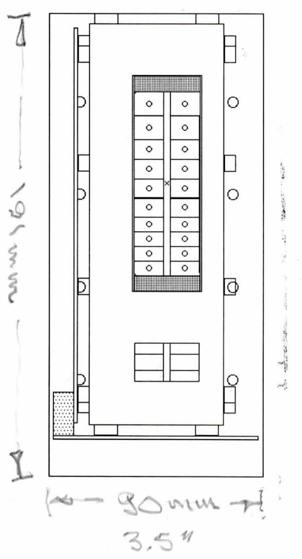
Shutter Board
CUSP 2001
UMD ION SPECTROGRA
4 DECEMBER 2000
J. H. MOORE



170 mm 6.625"
Telemetry Interfac Board on High Energ Side
Field-of-Vi
CUSP 2001
UMD ION SPECTROGRA
4 DECEMBER 2000
J. H. MOORE



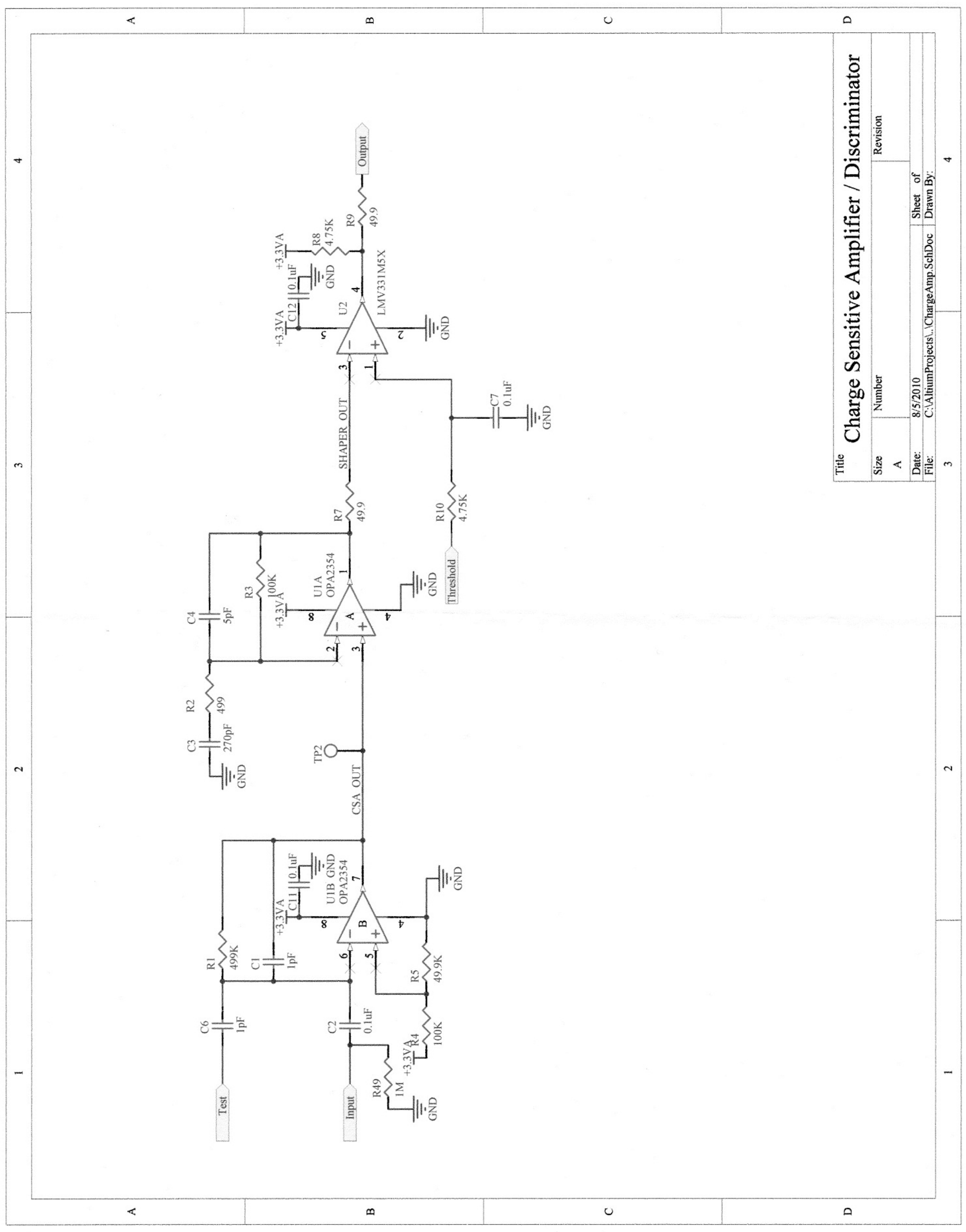
.12
CONNECTC
DEFLECTOR TERMINA
Telemetry Interfac Board on Low Energ Side
CUSP 2001
UMD ION SPECTROGRA
4 DECEMBER 2000
J. H. MOORE

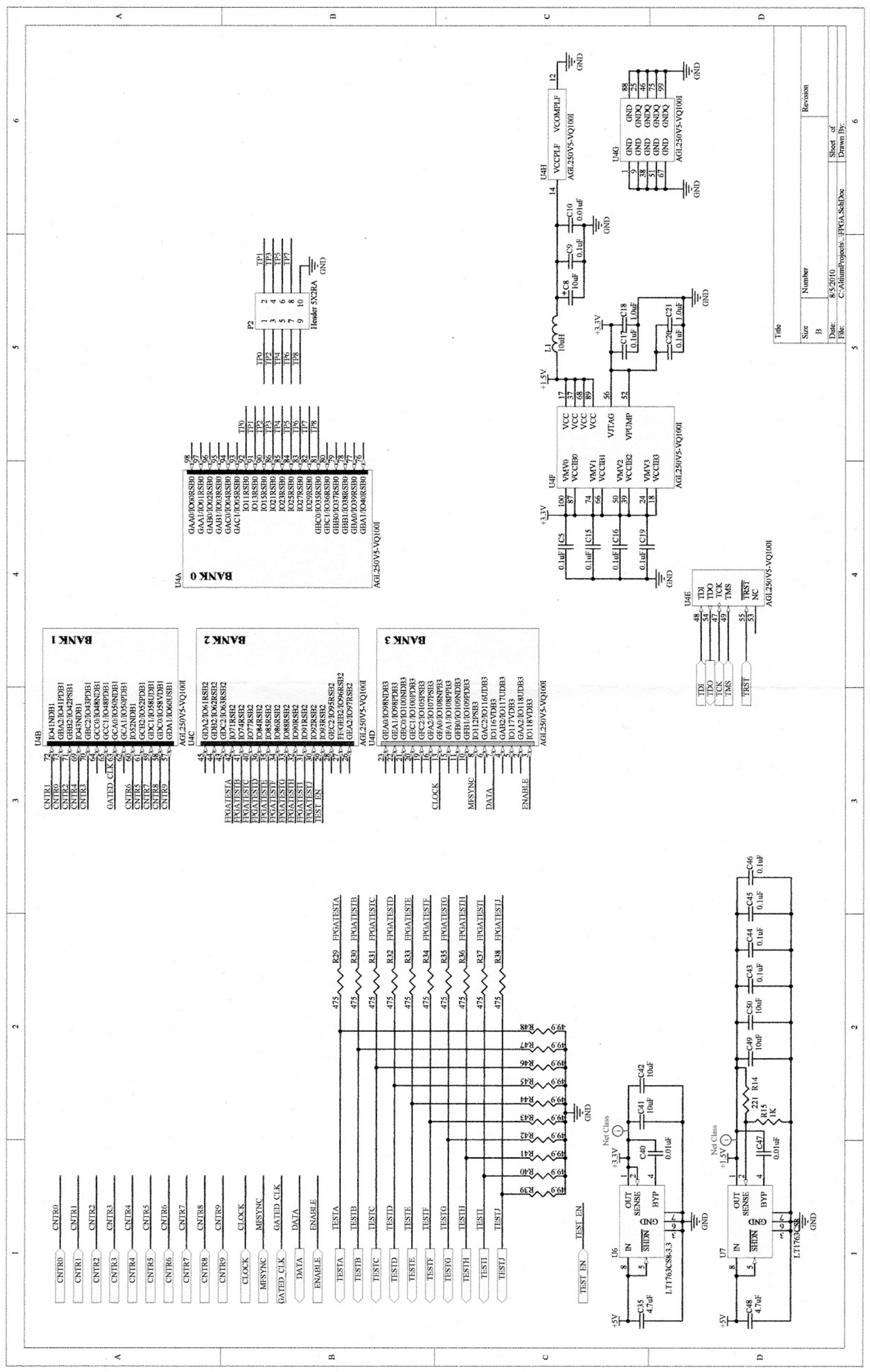


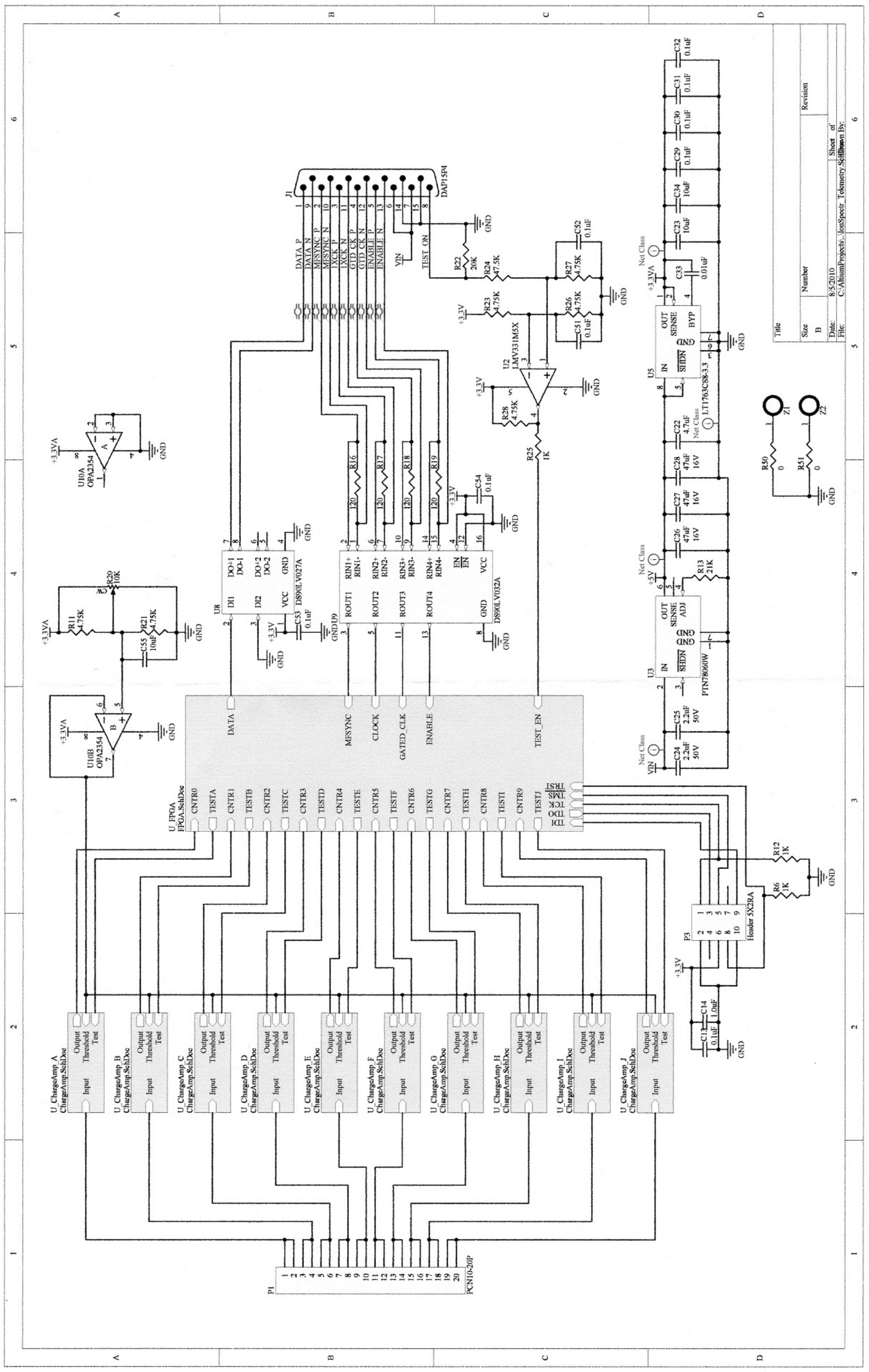
191 mm 7.5"
90 mm 3.5"

A.3 KIMS Electronics

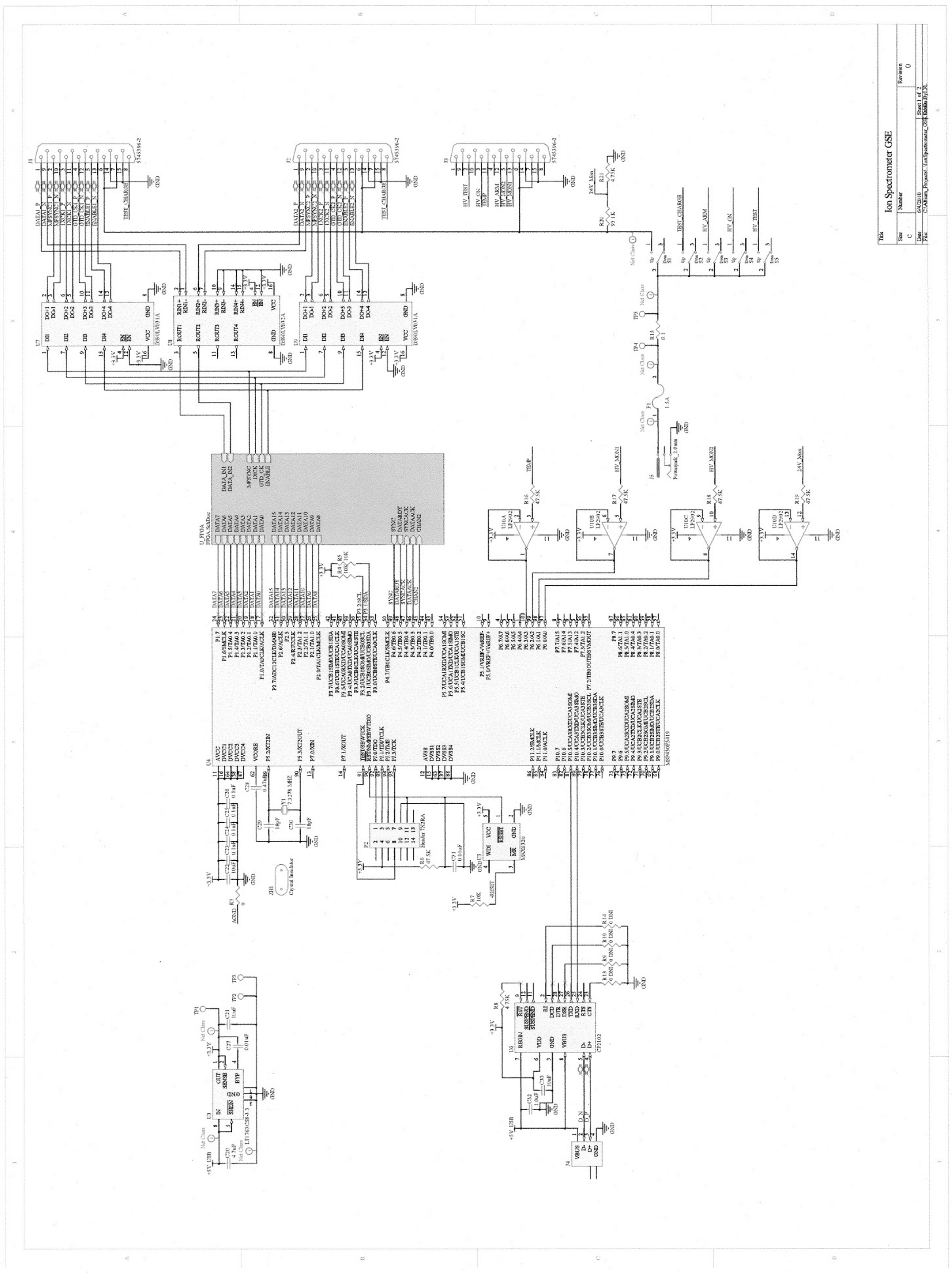
The following pages include the circuit schematics for the KIMS electronics.

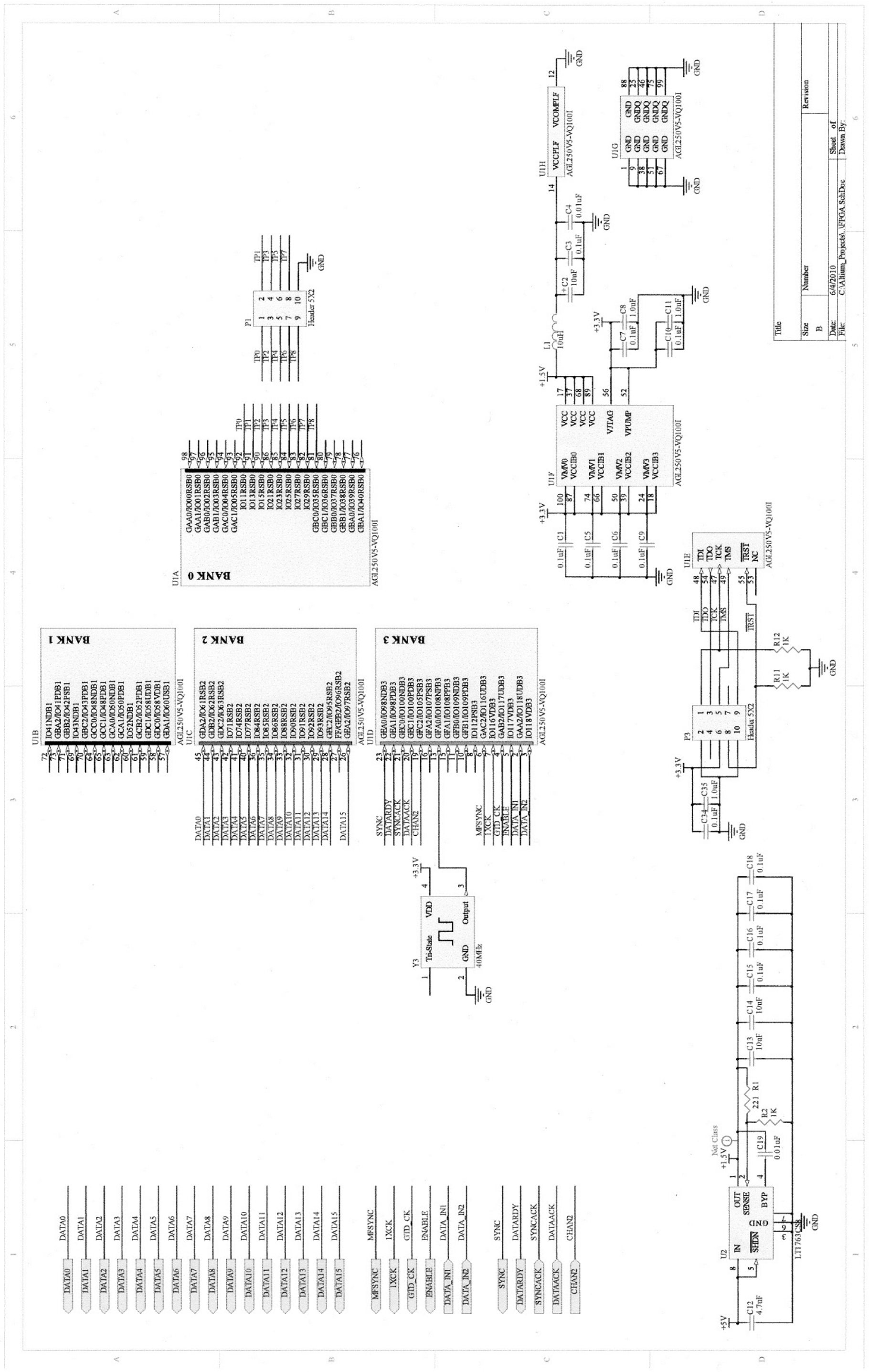

Charge Sensitive Amplifier / Discriminator
Title
Size
A
Number
Revision
Date: 8/5/2010
File: C:\AltiumProjects\..\ChargeAmp.SchDoc
Sheet of
Drawn By:
Test
Input
Output
Threshold
CSA OUT
SHAPER OUT
TP2
U1B OPA2354
U1A OPA2354
U2 LMV331M5X
C6 1pF
C1 1pF
C2 0.1uF
C11 0.1uF
C3 270pF
C4 5pF
C7 0.1uF
C12 0.1uF
R1 499K
R2 499
R3 100K
R4 100K
R5 49.9K
R7 49.9
R8 4.75K
R9 49.9
R10 4.75K
R49 1M
+3.3VA
GND


BANK 0
BANK 1
BANK 2
BANK 3
AGL250V5-VQ100I
Header 5X2RA
LT1763CS8-3.3
GND
+3.3V
+1.5V
+5V

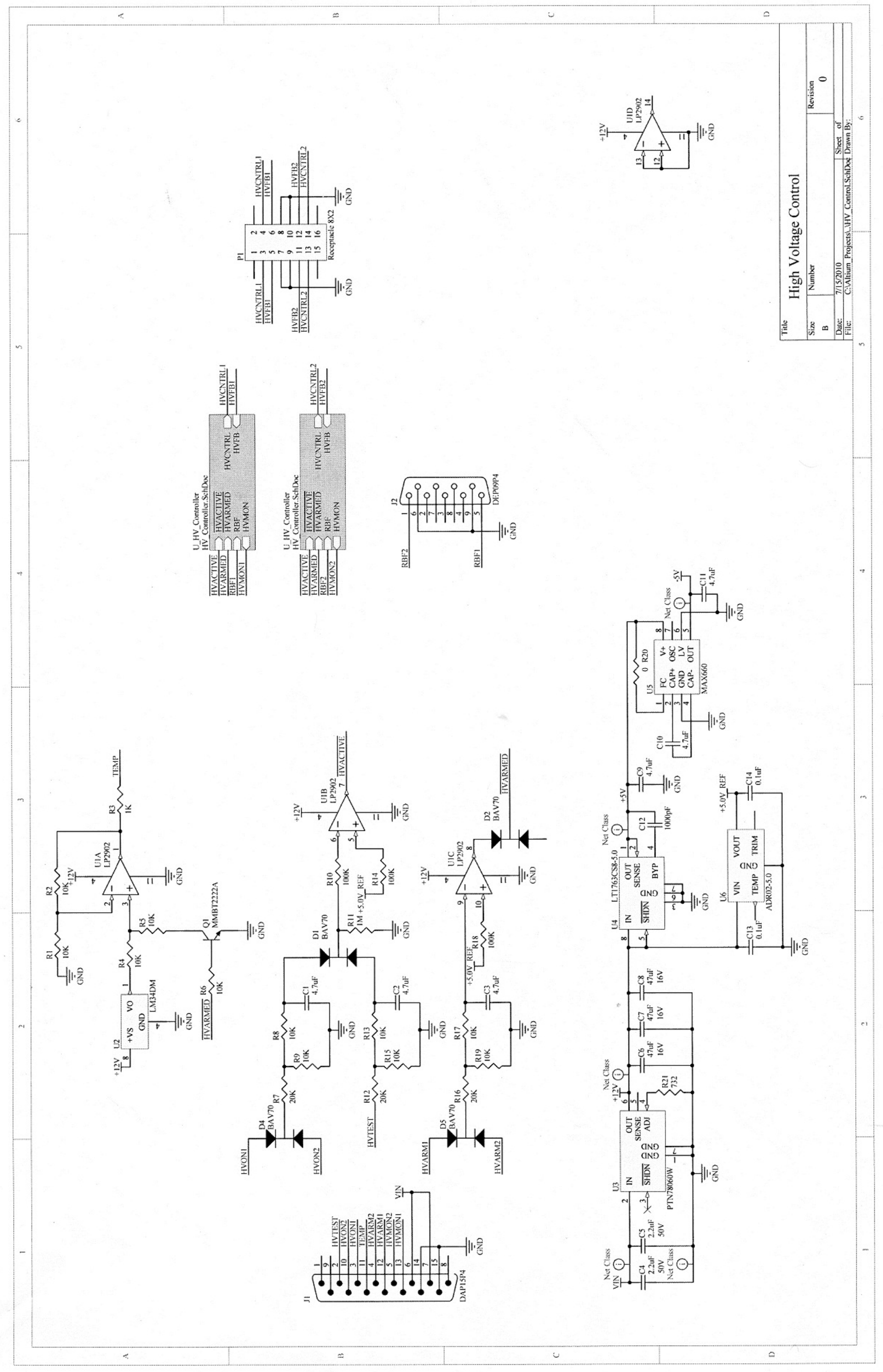


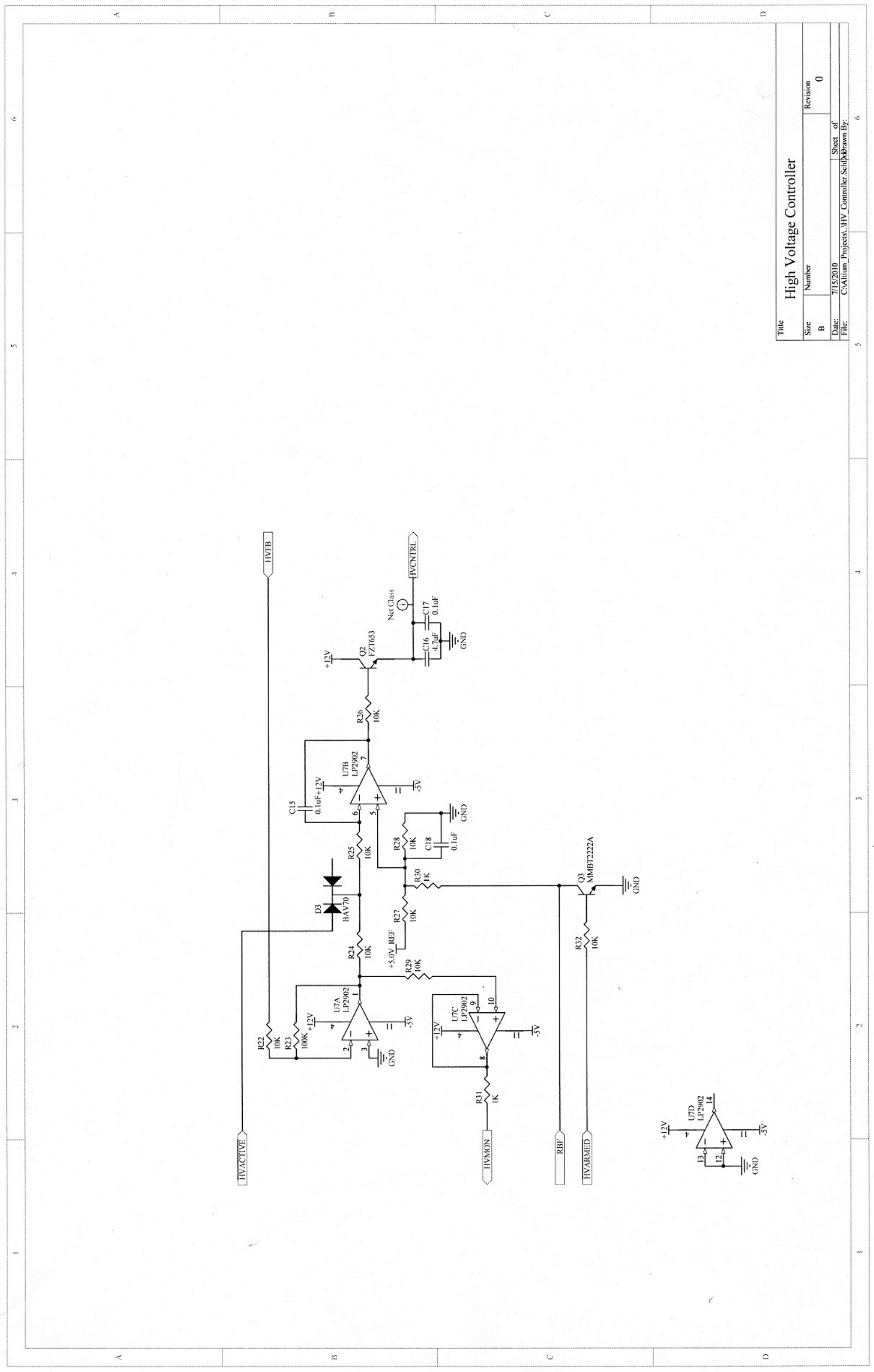
U_FPGA
FPGA.SchDoc
DATA
MFSYNC
CLOCK
GATED_CLK
ENABLE
TEST_EN
DS90LV027A
DS90LV032A
LMV331M5X
PTN78060W
LT1763CS8-3.3
OPA2354
DAP15P4
PCN10-20P
Header 5X2RA
GND
+3.3V
+3.3VA
+5V
VIN
Title
Size
Number
Revision
B
Date:
8-5-2010
File:
Sheet of
Drawn By:

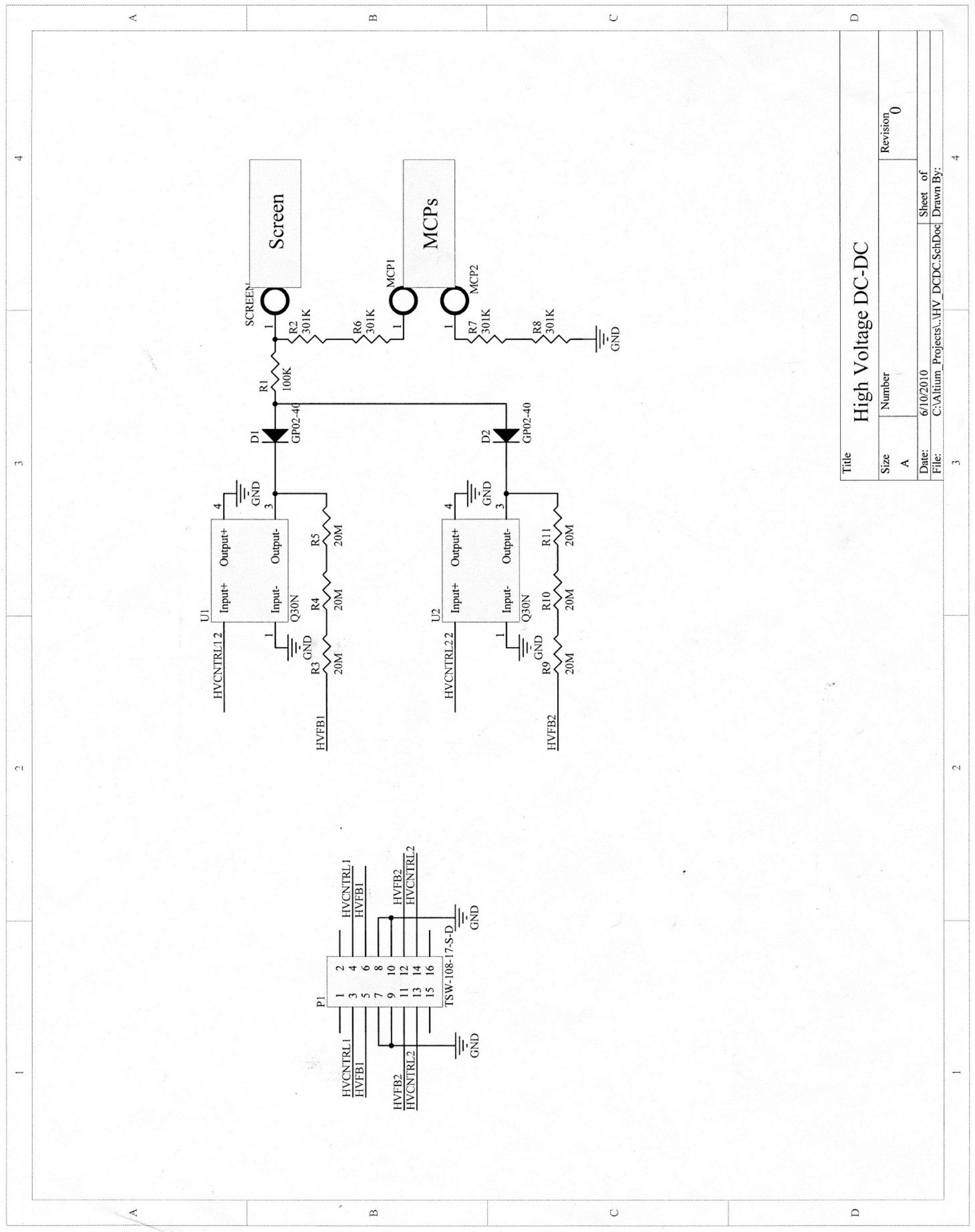

Title
Ion Spectrometer GSE
Size
C
Number
Revision
0
Date
6/4/2010
Sheet 1 of 2
File:
MSP430F5419
CP2102
FPGA_SlabDev
DS90LV031A
DS90LV032A
LP2902
J1
J2
J3
J4
J5
HV_TEST
HV_ON
HV_ARM
TEST_CHARGE
24V_Mon
HV_MON1
HV_MON2
TID-IP
GND
+3.3V
+5V_USB
Crystal Insulator
LT1763CS8-3.3
Header 7X2H
MAX6320



BANK 0
BANK 1
BANK 2
BANK 3
AGL250V5-VQ100I
Header 5X2
LT1763
40MHz
Title
Size
Number
Revision
B
Date: 6/4/2010
File: C:\Altium_Projects\...\FPGA.SchDoc
Sheet of
Drawn By:


Title
High Voltage Control
Size
B
Number
Revision
0
Date: 7/15/2010
Sheet of
File: C:\Altium_Projects\...\HV_Control.SchDoc
Drawn By:
LP2902
LM34DM
MMBT2222A
BAV70
LT1763CS8-5.0
PTN78060W
MAX660
ADR02-5.0
DAP15P4
DEP09P4
Receptacle 8X2
U_HV_Controller
HV_Controller.SchDoc


High Voltage Controller
Title
Size
B
Number
Revision
0
Date: 7/15/2010
Sheet of
File: C:\Altium_Projects_HV_Controller.SchDoc
Drawn By:
HVFB
HVCNTRL
Net Class
Q2
FZT653
+12V
C16
4.7uF
C17
0.1uF
GND
R26
10K
U7B
LP2902
C15
0.1uF
-5V
R25
10K
R28
10K
C18
0.1uF
D3
BAV70
R30
1K
Q3
MMBT2222A
R27
10K
+5.0V_REF
R24
10K
R29
10K
U7A
LP2902
R22
10K
R23
100K
U7C
LP2902
R31
1K
R32
10K
U7D
LP2902
HVACTIVE
HVMON
RBF
HVARMED
A
B
C
D
1
2
3
4
5
6



High Voltage DC-DC
Title
Size
A
Number
Revision
0
Date: 6/10/2010
File: C:\Altium_Projects\..\HV_DCDC.SchDoc
Sheet of
Drawn By:
Screen
MCPs
SCREEN
MCP1
MCP2
R1 100K
R2 301K
R6 301K
R7 301K
R8 301K
D1 GP02-40
D2 GP02-40
U1 Q30N
U2 Q30N
Input+
Input-
Output+
Output-
R3 20M
R4 20M
R5 20M
R9 20M
R10 20M
R11 20M
HVCNTRL1
HVCNTRL2
HVFB1
HVFB2
GND
P1
TSW-108-17-S-D

Appendix B

KIMS Software

B.1 Monte Carlo Simulation

B.1.1 mc_sim.py

```
from numpy import array,arange,append,shape,zeros,histogram
from pylab import loadtxt,plot,figure
from math import cos,sqrt,pi
from random import random

q  = 2*1.60218e-19
mp = 1.67262e-27
B0  = 0.096

cel = -2.286*0.0254                # collimator edges
cer = -1.651*0.0254

dhel = -1.130*0.0254               # detector housing edges
dher =  2.469*0.0254

pcb_w = 3.265*0.0254
pcb_h = 0.945*0.0254
mask  = 0.115*0.0254
partition = 0.118*0.0254
pad_height = (pcb_h-partition)/2. - mask

off  = (dhel + dher - pcb_w)/2.0     # offset to edge of pcb: pcb edge is 0
alpha = (cer + cel)/2.0 - off       # location of aperture center
w = 0.118*0.0254                   # aperture half width

N_pads=10
x=loadtxt(raw_input('Filename: '))

tm = -6.0*pi/180.
tp = 19.0*pi/180.
```

```
def locate(E,theta,s,m,B):
    return alpha + s + 2*sqrt(2*m*E/q)/B * cos(theta)

def pad_sim(N,E_min,E_max,theta_min,theta_max,s_min,s_max,a,b,m=mp,B=B0):
    hits = []
    done = 0.10
    for n in range(N):
        e = E_min + (E_max-E_min)*random()
        t = theta_min + (theta_max-theta_min)*random()
        s = s_min + (s_max-s_min)*random()
        pos = locate(e,t,s,m,B)
        if pos >= a and pos <= b:
            hits.append((e,t,s))
        if n > N*done:
            print done*100,'% done'
            done += 0.10
    print 100.0, '% done'
    out = zeros(shape(hits))
    I,J = shape(hits)
    for i in range(I):
        for j in range(J):
            out[i,j]=hits[i][j]
    return out

def KIMS_hist(E,a,b,m=mp,B=B0):
    hits=pad_sim(int(2e7),E[0],E[1],tm,tp,-w,w,a,b,m,B)
    dE=5.0
    edges=arange(E[0],E[1]+dE,dE)
    centers=zeros(len(edges)-1)
    for i in range(len(centers)):
        centers[i]=(edges[i]+edges[i+1])/2.0
    hist,junk=histogram(hits[:,0],bins=edges,normed=True)
    plot(centers,hist)
    return hits,edges,hist

def full_sim():

    E=array([0,0])
    E[0]=float(raw_input("Lowest energy in eV: "))
    E[1]=float(raw_input("Highest energy in eV: "))
    BA=float(raw_input("Low Energy B: "))
    BB=float(raw_input("High Energy B: "))
    m=float(raw_input("Particle Mass: "))

    B = BA
```

```
figure()
EA = zeros(10)
dEA = zeros(10)

print(":: A0 ::")
hits_a0,edges,hist_a0 = KIMS_hist(E,x[0],x[1],m,B)
centers=zeros(len(edges)-1)
for i in range(len(centers)):
    centers[i]=(edges[i]+edges[i+1])/2.0
EA[0] = sum(centers*hist_a0*5.0)
dEA[0] = 1./max(hist_a0)

print(":: A1 ::")
hits_a1,edges,hist_a1 = KIMS_hist(E,x[2],x[3],m,B)
EA[1] = sum(centers*hist_a1*5.0)
dEA[1] = 1./max(hist_a1)

print(":: A2 ::")
hits_a2,edges,hist_a2 = KIMS_hist(E,x[4],x[5],m,B)
EA[2] = sum(centers*hist_a2*5.0)
dEA[2] = 1./max(hist_a2)

print(":: A3 ::")
hits_a3,edges,hist_a3 = KIMS_hist(E,x[6],x[7],m,B)
EA[3] = sum(centers*hist_a3*5.0)
dEA[3] = 1./max(hist_a3)

print(":: A4 ::")
hits_a4,edges,hist_a4 = KIMS_hist(E,x[8],x[9],m,B)
EA[4] = sum(centers*hist_a4*5.0)
dEA[4] = 1./max(hist_a4)

print(":: A5 ::")
hits_a5,edges,hist_a5 = KIMS_hist(E,x[10],x[11],m,B)
EA[5] = sum(centers*hist_a5*5.0)
dEA[5] = 1./max(hist_a5)

print(":: A6 ::")
hits_a6,edges,hist_a6 = KIMS_hist(E,x[12],x[13],m,B)
EA[6] = sum(centers*hist_a6*5.0)
dEA[6] = 1./max(hist_a6)

print(":: A7 ::")
hits_a7,edges,hist_a7 = KIMS_hist(E,x[14],x[15],m,B)
EA[7] = sum(centers*hist_a7*5.0)
```

```
dEA[7] = 1./max(hist_a7)

print(":: A8 ::")
hits_a8,edges,hist_a8 = KIMS_hist(E,x[16],x[17],m,B)
EA[8] = sum(centers*hist_a8*5.0)
dEA[8] = 1./max(hist_a8)

print(":: A9 ::")
hits_a9,edges,hist_a9 = KIMS_hist(E,x[18],x[19],m,B)
EA[9] = sum(centers*hist_a9*5.0)
dEA[9] = 1./max(hist_a9)


B = BB
figure()
EB = zeros(10)
dEB = zeros(10)

print(":: B0  :")
hits_b0,edges,hist_b0 = KIMS_hist(E,x[0],x[1],m,B)
EB[0] = sum(centers*hist_b0*5.0)
dEB[0] = 1./max(hist_b0)

print(":: B1 ::")
hits_b1,edges,hist_b1 = KIMS_hist(E,x[2],x[3],m,B)
EB[1] = sum(centers*hist_b1*5.0)
dEB[1] = 1./max(hist_b1)

print(":: B2 ::")
hits_b2,edges,hist_b2 = KIMS_hist(E,x[4],x[5],m,B)
EB[2] = sum(centers*hist_b2*5.0)
dEB[2] = 1./max(hist_b2)

print(":: B3 ::")
hits_b3,edges,hist_b3 = KIMS_hist(E,x[6],x[7],m,B)
EB[3] = sum(centers*hist_b3*5.0)
dEB[3] = 1./max(hist_b3)

print(":: B4 ::")
hits_b4,edges,hist_b4 = KIMS_hist(E,x[8],x[9],m,B)
EB[4] = sum(centers*hist_b4*5.0)
dEB[4] = 1./max(hist_b4)

print(":: B5 ::")
hits_b5,edges,hist_b5 = KIMS_hist(E,x[10],x[11],m,B)
```

```
EB[5] = sum(centers*hist_b5*5.0)
dEB[5] = 1./max(hist_b5)

print(":: B6 ::")
hits_b6,edges,hist_b6 = KIMS_hist(E,x[12],x[13],m,B)
EB[6] = sum(centers*hist_b6*5.0)
dEB[6] = 1./max(hist_b6)

print(":: B7 ::")
hits_b7,edges,hist_b7 = KIMS_hist(E,x[14],x[15],m,B)
EB[7] = sum(centers*hist_b7*5.0)
dEB[7] = 1./max(hist_b7)

print(":: B8 ::")
hits_b8,edges,hist_b8 = KIMS_hist(E,x[16],x[17],m,B)
EB[8] = sum(centers*hist_b8*5.0)
dEB[8] = 1./max(hist_b8)

print(":: B9 ::")
hits_b9,edges,hist_b9 = KIMS_hist(E,x[18],x[19],m,B)
EB[9] = sum(centers*hist_b9*5.0)
dEB[9] = 1./max(hist_b9)

return EA,dEA,EB,dEB
```

B.2 FPGA

B.2.1 Telemetry/Amplifier Board

B.2.1.1 Telemetry Code

```
----------------------------------------------------------------------------
-- Company:         University of Maryland
-- Engineer:        Larry Lutz
--
-- Create Date:    01/17/2010
-- Module Name:    IonSpectr_TM - Behavioral
--
-- Revision:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity IonSpectr_TM is
    Port (  Clock : in STD_LOGIC;         -- Telemetry Clock Input (10 MHz)

            MFSync : in STD_LOGIC;        -- Major Frame Sync

            GtdClk : in STD_LOGIC;        -- Gated Clock

            Counter_Input : in  STD_LOGIC_VECTOR (9 downto 0);
                                          -- Counter Inputs from CSAs

            DataOut : out STD_LOGIC;      -- Data Output to Telemetry

            Test_On : in STD_LOGIC;       -- Test On Command

            Calibrate_Out : out STD_LOGIC_VECTOR (9 downto 0);
                                          -- Calibration Outputs

            TP : out STD_LOGIC_VECTOR (8 downto 0)
                                          -- Test Points
```

```
        );
end IonSpectr_TM;

architecture Behavioral of IonSpectr_TM is



component Counter
port (
        Clock : in  STD_LOGIC;

        Reset : in STD_LOGIC;

        Counter_In : in STD_LOGIC;
        Dataout : out  STD_LOGIC_VECTOR (19 downto 0)
);
end component;



signal Shiftreg : std_logic_vector (199 downto 0);
                                          -- Shift Register

signal Shiftin : std_logic_vector (199 downto 0);
                                          -- Shift Register Input

signal CalCount : std_logic_vector (14 downto 0);
                                          -- Calibration Counter/Divider

signal MFCount : std_logic_vector (2 downto 0);
                                          -- MFSync Timing Counter

signal Reset : std_logic;                 -- Counter Reset

signal Srload : std_logic;                -- Shift Register Load





begin



process (Clock, MFSync, MFCount)          -- MFSync Timing Counter
```

```
begin
    if MFSync = '0' then MFCount <= "000";
    elsif rising_edge (Clock) then
        if MFCount /= "111" then
            MFCount <= MFCount + '1';
        end if;
    end if;
end process;


Srload <= '1' when MFCount = "001" else '0';
Reset <= '1' when MFCount = "011" else '0';



process (Srload, GtdClk, Shiftreg, Shiftin)
                                          -- Shift Register
begin
    if Srload = '1' then Shiftreg <= Shiftin;
    elsif rising_edge (GtdClk) then
        Shiftreg <= Shiftreg (198 downto 0) & '0';
    end if;
end process;


DataOut <= Shiftreg (199);
```

```
process (Clock, Test_On, CalCount)        -- Calibration Counter/Divider

begin

    if Test_On = '0' then CalCount <= "000000000000000";

    elsif rising_edge (Clock) then

        CalCount <= CalCount + '1';

    end if;

end process;


Calibrate_Out <= CalCount (14 downto 5);




TP(0) <= GtdClk;

TP(1) <= Shiftreg (199);

TP(2) <= Clock;

TP(3) <= Counter_Input (0);

TP (8 downto 4) <= "00000";



Counters:                                 -- Generate Counters
   for I in 0 to 9 generate
   begin
CounterGen : Counter
port map (
    Clock => Clock,

    Reset => Reset,
    --Counter_In => CalCount (I+5),

    Counter_In => Counter_Input (I),
    Dataout => Shiftin (((I*20)+19) downto (I*20))
);
   end generate;



end Behavioral;
```

B.2.1.2 Amplifier/Counter Code

```
----------------------------------------------------------------------------
-- Company:          University of Maryland
-- Engineer:         Larry Lutz
--
-- Create Date:    01/17/2010
-- Module Name:    Counter - Behavioral
--
-- Revision:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity Counter is
    Port (  Clock : in STD_LOGIC;         -- Clock Input

            Reset : in STD_LOGIC;         -- Reset Input

            Counter_In : in STD_LOGIC;    -- Counter Input

            DataOut : out STD_LOGIC_VECTOR (19 downto 0)
                                          -- Data Output

          );
end Counter;

architecture Behavioral of Counter is



signal Count : std_logic_vector (15 downto 0);
                                          -- Count Value

signal InputReg : std_logic;              -- Input Registered

signal InputEdge : std_logic;             -- Input Edge
```

```
begin


process (Clock, InputReg, Counter_In)     -- Input Edge Detector

begin

    if rising_edge (Clock) then

        InputReg <= Counter_In;

    end if;

end process;


InputEdge <= ’1’ when InputReg = ’1’ and Counter_In = ’0’ else ’0’;


process (Reset, Clock, InputEdge, Count)  -- Counter

begin

    if Reset = ’1’ then Count <= "0000000000000000";

    elsif rising_edge (Clock) then

        if InputEdge = ’1’ then

            Count <= Count + ’1’;

        end if;

    end if;

end process;


DataOut (19 downto 16) <= "0000";

DataOut (15 downto 0) <= Count;
```

```
--DataOut <= "10100000000000000000";


end Behavioral;
```

B.2.2 GSE

```
----------------------------------------------------------------------------
-- Company:         University of Maryland
-- Engineer:        Larry Lutz
--
-- Create Date:    05/31/2010
-- Module Name:    IonSpectr_GSE - Behavioral
--
-- Revision:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity IonSpectr_GSE is
    Port (  Osc : in STD_LOGIC;           -- Oscillator (40 MHz)

    -- TM Interface

            ClockOut : out STD_LOGIC;     -- Clock Out (10 MHz)

            MFSync : out STD_LOGIC;       -- Major Frame Sync

            GatedClock : out STD_LOGIC;   -- Gated Clock

            Enable : out STD_LOGIC;       -- Enable

            DataIn1 : in STD_LOGIC;       -- Data Input Side 1

            DataIn2 : STD_LOGIC;          -- Data Input Side 2

    -- MicroController Interface

            DataOut : out STD_LOGIC_VECTOR (15 downto 0);
                                          -- Data from Shift Registers

            Sync : out STD_LOGIC;         -- Frame Sync

            SyncAck : in STD_LOGIC;       -- Sync Acknowledge
            DataRdy : out STD_LOGIC;      -- Data Ready

            DataAck : in STD_LOGIC;       -- Data Acknowledge

            Chan2 : in STD_LOGIC;         -- Channel 2 Data Select

    -- Test Points

            TP : out STD_LOGIC_VECTOR (8 downto 0)
                                          -- Test Points

          );
end IonSpectr_GSE;

architecture Behavioral of IonSpectr_GSE is



signal Clockdiv : std_logic_vector (1 downto 0);
                                          -- Clock Divider

signal Divide20 : std_logic_vector (4 downto 0);
                                          -- 20 Bit Word Divider

signal WordCnt  : std_logic_vector (12 downto 0);
                                          -- Word Counter

signal GateCnt  : std_logic_vector (4 downto 0);
                                          -- Gate Counter

signal DataReg1 : std_logic_vector (15 downto 0);
                                          -- Data 1 Shift Register

signal DataReg2 : std_logic_vector (15 downto 0);
                                          -- Data 2 Shift Register

signal DataInt  : std_logic;              -- Data Interval

signal Word20   : std_logic;              -- Word Synch Bit

signal Clock10  : std_logic;              -- 10 MHz Synch Bit

signal Frame    : std_logic;              -- Frame Sync

signal GateCtEn : std_logic;              -- Gate Counter Enable
signal Gate     : std_logic;             -- Gate for Gated Clock

signal GtdClock : std_logic;             -- Gated Clock

signal GatCkEdg : std_logic;             -- Gated Clock Edge

signal DataMicr : std_logic;             -- Data Ready for MicroController


constant Count20 : std_logic_vector  := "10011";
                                         -- 20 Bit Word End Count (19)



begin


process (Osc, Clockdiv)                  -- Clock Divider for 10 MHz

begin

    if rising_edge (Osc) then

        Clockdiv <= Clockdiv + '1';

    end if;

end process;


Clock10 <= '1' when Clockdiv = "00" else '0';

ClockOut <= Clockdiv(1);




process (Osc, Clock10, Word20, Divide20)  -- Clock Divider for 20 Bit Words
```

```
begin
    if rising_edge (Osc) then
        if Clock10 = '1' then
            if Word20 = '1' then Divide20 <= "00000";
            else Divide20 <= Divide20 + '1';
            end if;
        end if;
    end if;
end process;

Word20 <= '1' when Divide20 = Count20 else '0';

process (Osc, Clock10, Word20, WordCnt)   -- Word Counter
begin
    if rising_edge (Osc) then
        if Clock10 = '1' and Word20 = '1' then
        WordCnt <= WordCnt + '1';
        end if;
    end if;
end process;

Frame <= '1' when WordCnt (12 downto 6) = "00000000" else '0';
MFSync <= Frame;
```

```
Sync <= Frame;

DataInt <= '1' when WordCnt (8 downto 6) = "00000" else '0';

DataMicr <= DataInt and not Gate and not Frame;

DataRdy <= DataMicr;


process (Osc, DataInt, Clock10, GateCtEn, GateCnt)
                                          -- Gate Clock Divider (20 Bits)

begin

    if DataInt = '0' then GateCnt <= "00000";

    elsif rising_edge (Osc) then

        if Clock10 = '1' and GateCtEn = '1' then

            GateCnt <= GateCnt + '1';

        end if;

    end if;

end process;


GateCtEn <= '0' when GateCnt = "10100" else '1';

Gate <= DataInt and GateCtEn and not Frame;

Enable <= Gate;

GtdClock <= Gate and Clockdiv(1);

GatedClock <= GtdClock;


GatCkEdg <= '1' when GtdClock = '1' and Clockdiv = "10" else '0';
```

```
process (Osc, GatCkEdg, DataReg1, DataIn1)
                                          -- Data 1 Shift Register
begin
   if rising_edge (Osc) then
      if GatCkEdg = '1' then
         DataReg1 <= DataReg1(14 downto 0) & DataIn1;
      end if;
   end if;
end process;



process (Osc, GatCkEdg, DataReg2, DataIn2)
                                          -- Data 2 Shift Register
begin
   if rising_edge (Osc) then
      if GatCkEdg = '1' then
         DataReg2 <= DataReg2(14 downto 0) & DataIn2;
      end if;
   end if;
end process;



DataOut <= DataReg1 when Chan2 = '0' else DataReg2;





TP(0) <= Frame;

TP(1) <= DataMicr;

TP(2) <= GtdClock;

TP(3) <= GatCkEdg;

TP(4) <= SyncAck;
TP(5) <= DataAck;

TP(6) <= DataIn1;

TP(7) <= DataIn2;

TP(8) <= '0';


end Behavioral;
```

B.3 MSP430

The following code programmed the MSP430F5438 on the GSE board to interface with the KIMS telemetry boards. The MSP430 served to send the data received from the telemetry boards by an FPGA to a computer where they could be recorded to a file.

```c
/* KIMSF5438 v. 0.5:  24 August 2010
 * GSE programming for the MSP430F5438.
 * Increase operating/transmission rates to see if it affects data read.
 */

#include <msp430f5438.h>
#include <stdio.h>
#include <string.h>

#define CHAN2   BIT2
#define DATAACK BIT3
#define SYNCACK BIT4
#define DATARDY BIT5
#define SYNC    BIT6

#define V24_MON BIT0
#define HV_MON2 BIT1
#define HV_MON1 BIT2
#define TEMP    BIT3

void port_init(void);
void xtal_init(void);
void uart_init(void);
void adc12_init(void);

void call_0(void);
void call_1(void);
void call_2(void);
void call_3(void);
void call_r(void);
void send_message(void);

int analog[4];
int ad_ready;
```

```
char command = 'p';

void main(void) {
    int i, j, data_value, count;    // Two iterators and storage for data
    char hex_word[5];               // String for converting data to ASCII hex
    char message[36];

    WDTCTL = WDTPW + WDTHOLD;       // WDT off

    port_init();
    xtal_init();
    uart_init();
    adc12_init();
    _enable_interrupts();           // Allow interrupts without entering LPM

    i=0; j=0; ad_ready=0; count=0;  // initialize software flags/iterators

    for (;;) {
        switch (command) {
            case 'p':
                break;              // hold until command issued
            case 's':
                sprintf(message, "P4IN: %d\tSYNC: %d\tP4SY:%d\r\n", P4IN,
                        SYNC, (P4IN & SYNC));
                j=0;
                while (message[j] !='\0') {
                    while (!(UCA3IFG&UCTXIFG));
                    UCA3TXBUF = message[j++];
                }
                command='p';
                break;
            case 'r':
            while (count++ < 100) {
                while ((P4IN & SYNC) == SYNC);  // Pass any missed frames
                while ((P4IN & SYNC) != SYNC);  // Wait for MFRAME SYNC
                P4OUT |= BIT4;                  // Acknowledge SYNC

                for (i=0; i<10; i++) {      // 10 sets of data

                    while ((P4IN & DATARDY) != DATARDY);    // Wait for DATARDY
                    P4OUT |= DATAACK;       // Acknowledge DATARDY

                    data_value = PAIN;      // Read KIMS_A word
                    sprintf(hex_word, "%04x\t", data_value);
                    for (j=0; j<5; j++) {   // Send KIMS_A word
```

```
                        while (!(UCA3IFG & UCTXIFG));   // Wait for TX finish
                        UCA3TXBUF = hex_word[j];
                    }

/*                  data_value = P2IN;
                    sprintf(hex_word, "%02x\t", data_value);
                    for (j=0; j<3; j++) {
                        while (!(UCA3IFG & UCTXIFG));
                        UCA3TXBUF = hex_word[j];
                    }
                    data_value = P1IN;
                    sprintf(hex_word, "%02x\t", data_value);
                    for (j=0; j<3; j++) {
                        while (!(UCA3IFG & UCTXIFG));
                        UCA3TXBUF = hex_word[j];
                    }

*/                  P4OUT |= CHAN2;           // Switch to KIMS_B (CHAN2 on)
                    __delay_cycles(1000);     // Wait for data to settle

                    data_value = PAIN;        // Read KIMS_B word
                    sprintf(hex_word, "%04x\t", data_value);
                    for (j=0; j<5; j++) {     // Send KIMS_B word
                        while (!(UCA3IFG & UCTXIFG));   // Wait for TX finish
                        UCA3TXBUF = hex_word[j];
                    }

                    P4OUT &= ~CHAN2;          // Switch to KIMS_A (CHAN2 off)
                    P4OUT &= ~DATAACK;        // Data received, wait for next
                                              // DATARDY
                }

                P4OUT &= ~SYNCACK;            // Sample received, wait for next
                                              // MFRAME SYNC

                ADC12CTL0 |= ADC12SC;         // Start ADC conversions
                while (!(ad_ready));          // wait for ADC data
                for (i=0; i<4; i++) {         // Send Analog words
                    sprintf(hex_word, "%04x\t", analog[i]);
                    for (j=0; j<5; j++) {         // Send V24_MON word
                        while (!(UCA3IFG & UCTXIFG));   // Wait for TX finish
                        UCA3TXBUF = hex_word[j];
                    }
                }
                ad_ready = 0;                 // reset software flag
                while (!(UCA3IFG & UCTXIFG));   // Done with data, send newline
                UCA3TXBUF = '\r';
                while (!(UCA3IFG & UCTXIFG));
                UCA3TXBUF = '\n';
            }
                count = 0;
                command = 'p';
                break;
            default:
                sprintf(message, "Unknown command %c-- holding.\r\n", command);
                j=0;
                while (message[j] != '\0') {
                    while (!(UCA3IFG & UCTXIFG));
                    UCA3TXBUF = message[j++];
                }
                command = 'p';
                break;
        } // switch
    } // for(;;)
} // main

void port_init(void) {
    P1DIR = 0x00;                       // Ports 1 and 2 are data in (Port A word)
    P2DIR = 0x00;
    P3OUT = 0x06;                       // Set P3.1 and P3.2 on, all others off
    P3DIR = 0xFF;
    P4OUT = 0x00;                       // P4.0,1,7 unused, P4.2,3,4 initially off
    P4DIR = 0x9F;                       // P4.5,6 inputs, all others out.
    P5OUT = 0x00;
    P5DIR = 0xFF;                       // P5.2,3 already set to XT2
    P6OUT = 0x00;
    P6DIR = 0xF0;                       // P6.0-3 are ADC12, all others off
    P7OUT = 0x00;
    P7DIR = 0xFF;                       // All other ports off
    P8OUT = 0x00;
    P8DIR = 0xFF;
    P9OUT = 0x00;
    P9DIR = 0xFF;
    P10OUT = 0x00;
    P10DIR = 0xFF;
} // port_init

void xtal_init(void) {
    /* LFXT1 is the default source for ACLK and FLL.  These sources must be
     * changed in order to prevent XT1 fault flags from interrupting the code.
     * FLL is changed to source from XT2 and ACLK from REFO.  When the
     * oscillators are considered stable, ACLK is changed to source from XT2.
     */

    P5SEL |= 0x0c;                  // Port 5 select XT2
    UCSCTL6 &= ~XT2OFF;             // Enable XT2
    UCSCTL3 |= SELREF_2;            // Source FLL with XT2 since XT1 is not
                                    // present in the circuit.
    UCSCTL4 |= SELA_2;              // ACLK = REFO, SMCLK = DCO, MCLK = DCO

    __bis_SR_register(SCG0);        // Disable the FLL control loop
    UCSCTL0 = 0x0000;               // Set lowest possible DCOx, MODx
    UCSCTL1 = DCORSEL_5;            // Select DCO range 16MHz operation
    UCSCTL2 = FLLD_1 + 374;         // Set DCO Multiplier for 8MHz
                                    // (N + 1) * FLLRef = Fdco
                                    // (249 + 1) * 32768 = 8MHz
                                    // Set FLL Div = fDCOCLK/2
    __bic_SR_register(SCG0);        // Enable the FLL control loop

    // Worst-case settling time for the DCO when the DCO range bits have been
    // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in
    // 5xx User's Guide for optimization.
    // 32 x 32 x 8 MHz / 32,768 Hz = 250000 = MCLK cycles for DCO to settle
    __delay_cycles(375000);

    // Loop until Oscillators stabilize
    do {
                                    // Clear XT1, XT2, DCO fault flags
        UCSCTL7 &= ~(XT1LFOFFG + XT1HFOFFG + XT2OFFG + DCOFFG);
        SFRIFG1 &= ~OFIFG;          // Clear osc. fault interrupt flag
    } while (SFRIFG1 & OFIFG);      // Check osc. fault interrupt flag

    UCSCTL6 &= ~XT2DRIVE0;          // Drive for 7.3728 MHz
    UCSCTL4 |= SELA_5;              // ACLK = XT2
} // xtal_init

void uart_init(void) {
    P10SEL = 0x30;                  // Port 10 select USCI_A3 TxD/RxD
    UCA3CTL1 |= UCSWRST;            // Reset USCI state machine
    UCA3CTL1 |= UCSSEL_1;           // USCI CLK = ACLK (sourced by XT2)
    UCA3BR0 = 0x10;                 // 7.3728 MHz / 230400 = 32 -> 0x20
    UCA3BR1 = 0x00;                 // 0x0020 gives 230400 baud
                                    // 0x0080 gives 57600 baud
                                    // 0x0300 gives 9600 baud
    UCA3MCTL = UCBRS_0 + UCBRF_0;   // No modulation needed at this frequency
    UCA3CTL1 &= ~UCSWRST;            // Initialize USCI state machine
    UCA3IE |= UCRXIE;                // Enable USCI_A3 Rx interrupt.
} // uart_init

void adc12_init(void) {
    P6SEL |= 0x0F;                   // P6.0-3 ADC select
    ADC12CTL0 = ADC12MSC + ADC12SHT0_2 + ADC12ON;    // Sampling time, ADC12 on
    ADC12CTL1 = ADC12SHP + ADC12CONSEQ_1;   // Use sampling timer
    ADC12MCTL0 = ADC12INCH_0;        // channel A0 = V24_MON
    ADC12MCTL1 = ADC12INCH_1;        // channel A1 = HV_MON2
    ADC12MCTL2 = ADC12INCH_2;        // channel A2 = HV_MON1
    ADC12MCTL3 = ADC12INCH_3+ADC12EOS;   // channel A3 = TEMP, End of sequence
    ADC12IE = 0x08;                  // ADC12 interrupt flags data ready:
                                     // ADC12IFG.3
    ADC12CTL0 |= ADC12ENC;           // Enable ADC conversion
} // adc12_init

// Interrupt Service for UCA3 Rx interrupt
#pragma vector=USCI_A3_VECTOR
__interrupt void USCI_A3_ISR(void) {
    switch(__even_in_range(UCA3IV,4)) {
        case 0: break;               // Vector 0 - no interrupt
        case 2:                      // Vector 2 - RXIFG
            command = UCA3RXBUF;
            break;
        case 4:                      // Vector 4 - TXIFG
            break;
        default: break;
    }
} // USCI_A3_ISR

#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR (void)
{
    switch(__even_in_range(ADC12IV,34))
  {
  case  0: break;                            // Vector  0:  No interrupt
  case  2: break;                            // Vector  2:  ADC overflow
  case  4: break;                            // Vector  4:  ADC timing overflow
  case  6: break;                            // Vector  6:  ADC12IFG0
  case  8: break;                            // Vector  8:  ADC12IFG1
  case 10: break;                            // Vector 10:  ADC12IFG2
  case 12:                                   // Vector 12:  ADC12IFG3
    analog[0] = ADC12MEM0;                   // Move results, IFG is cleared
    analog[1] = ADC12MEM1;                   // Move results, IFG is cleared
    analog[2] = ADC12MEM2;                  // Move results, IFG is cleared
    analog[3] = ADC12MEM3;                  // Move results, IFG is cleared
    ad_ready = 1;
    break;
  case 14: break;                           // Vector 14:  ADC12IFG4
  case 16: break;                           // Vector 16:  ADC12IFG5
  case 18: break;                           // Vector 18:  ADC12IFG6
  case 20: break;                           // Vector 20:  ADC12IFG7
  case 22: break;                           // Vector 22:  ADC12IFG8
  case 24: break;                           // Vector 24:  ADC12IFG9
  case 26: break;                           // Vector 26:  ADC12IFG10
  case 28: break;                           // Vector 28:  ADC12IFG11
  case 30: break;                           // Vector 30:  ADC12IFG12
  case 32: break;                           // Vector 32:  ADC12IFG13
  case 34: break;                           // Vector 34:  ADC12IFG14
  default: break;
  }
}
```

B.4 Serial Communication

Communication with the GSE is done through a CP2102 Serial USB interface. This chip allows you to use the USB port of a computer as a serial port, from which you can communicate with the MSP430 via any serial terminal emulator. The open source software RealTerm [56] was used in this study.

Communication requires proper setup of the terminal. The MSP430 UART uses 8N1 (8 data bits, no parity, 1 stop bit), the common standard in UART transmission. The baud rate is set in the MSP430 code with the assignments to the registers UCA3BR0 and UCA3BR1. These registers give the number of clock cycles to divide to produce the correct timing for serial communication. With the crystal oscillator used, a division of 768 is required to get a baud rate of 9600. (ie. $7372800/768 = 9600$) In order to accommodate the faster rates of the telemetry board, faster baud rates are required. The code presented in the previous section sets the division factor to 0x0010, which provides a rate of 460800 baud. This rate should be set in the terminal accordingly.

Within the terminal, commands can be sent and data received. The valid commands in this version of the MSP430 code are as follows:

p	Hold and wait for another command. (Default mode)
s	Print a status message (used for debugging). Currently set to show the values in key registers to demonstrate timings are correct.
r	Read the next 100 data values received.
	Anything else returns an error to the terminal window and sets the mode to hold.

Table B.1: Description of commands available in the KIMS GSE.

Appendix C

KIMS Project Images

Assembly Photos

C.1 Magnetic Circuit Assembly

The following pages include photographs taken during the magnetic circuit assembly. See the checklist in Appendix A.1 for reference.

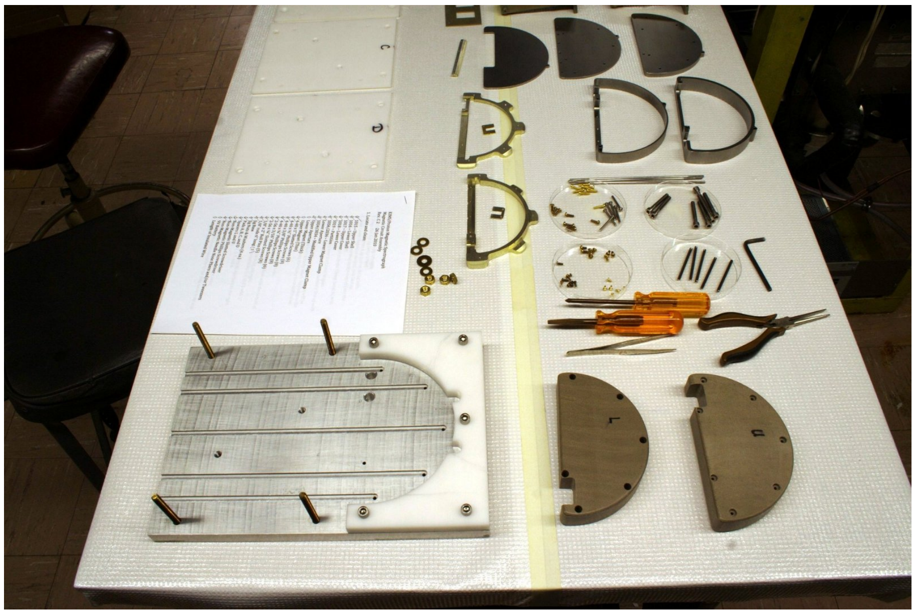


Figure C.1: Gathered parts for the magnetic circuit assembly.

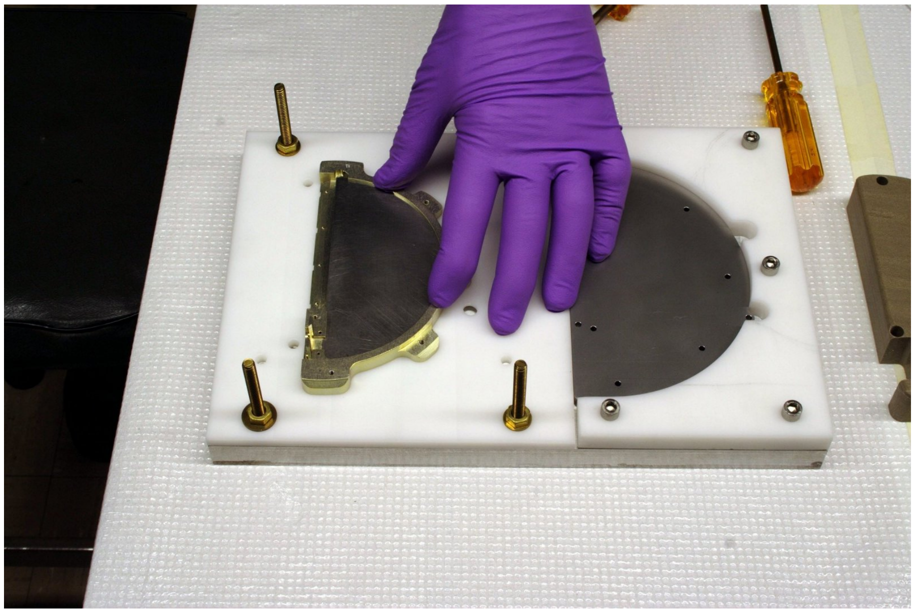


Figure C.2: Clamp placed on low field magnet. The magnet is slid onto the upper cover.

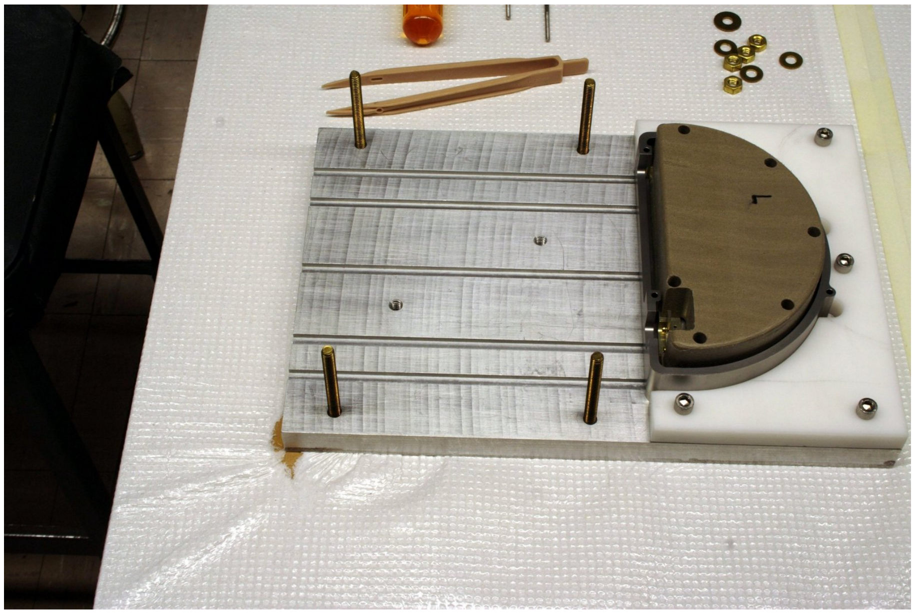


Figure C.3: The upper half of the magnetic circuit fully assembled. The nylon spacer was used to help keep alignment.

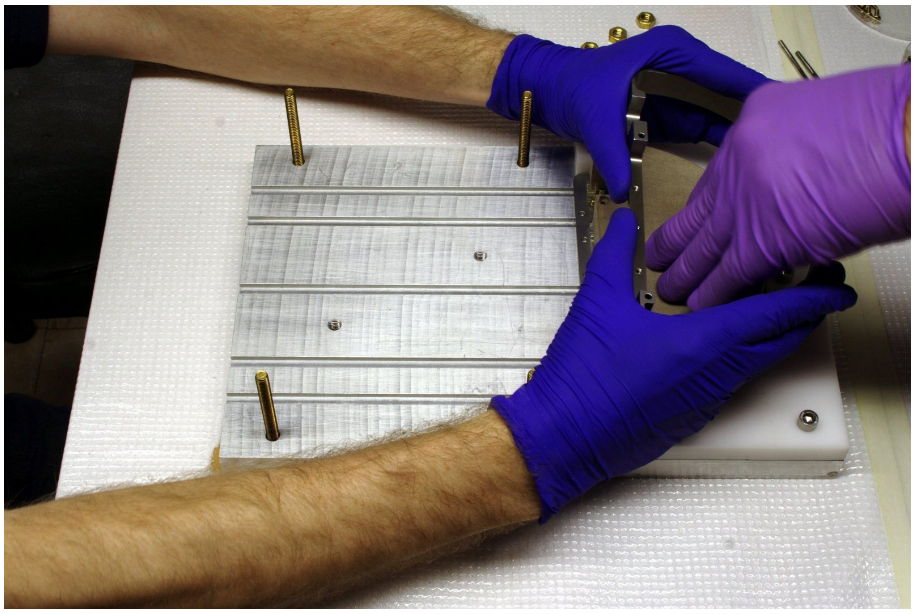


Figure C.4: Placing the lower shell around the magnet. A nylon spacer is used to help keep alignment.

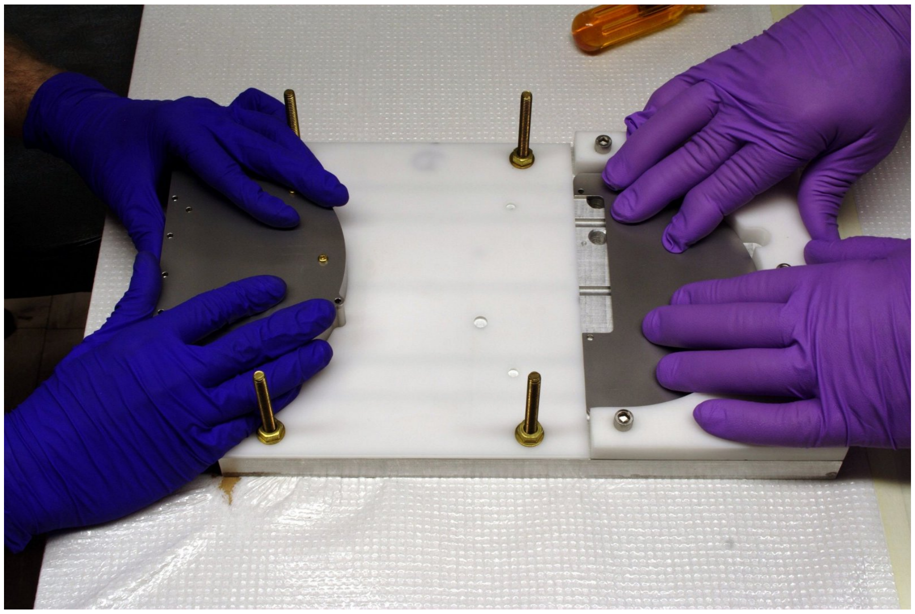


Figure C.5: The upper assembly is slid onto the partition.

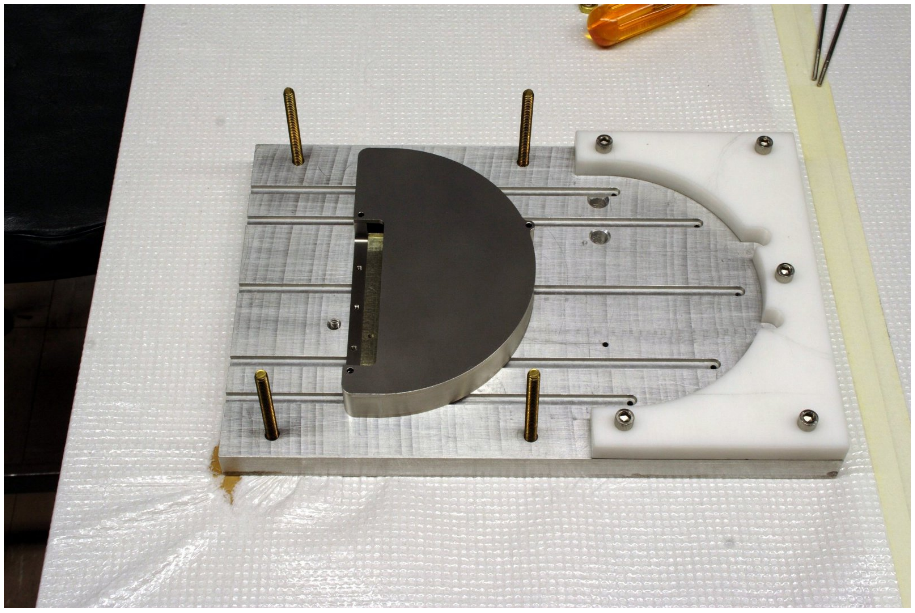


Figure C.6: Half of the magnetic circuit fully assembled. The final step is to slide the lower assembly onto the partition.

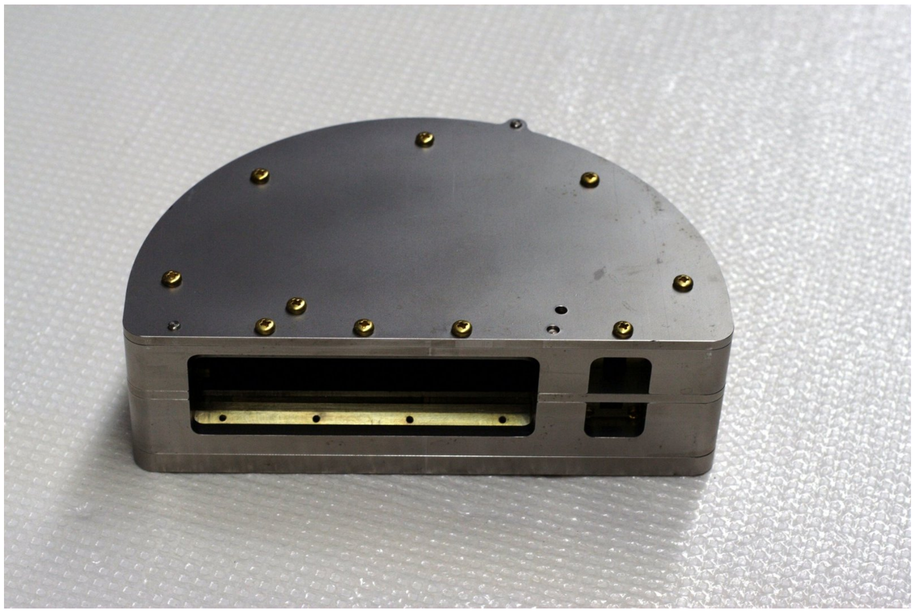


Figure C.7: The completed magnetic circuit. Stainless steel roll pins are used to align the parts. Brass screws secure the clamps to keep the magnets from shifting on launch.

C.2 KIMS Assembly

The following pages include photographs of the assembly of the KIMS instrument in preparation for vibration testing.

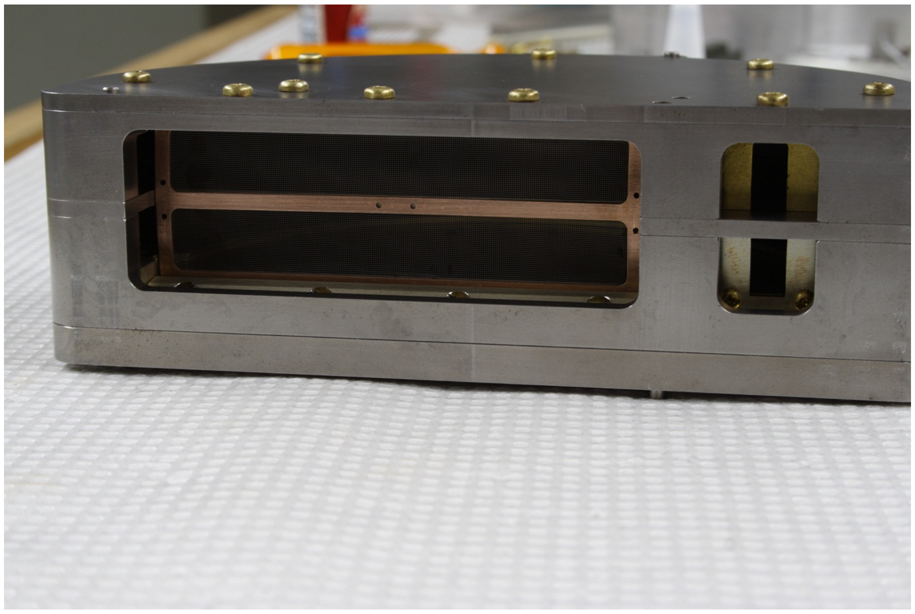


Figure C.8: The ground screen is placed inside the magnetic circuit and held in place with a compressive clamp.

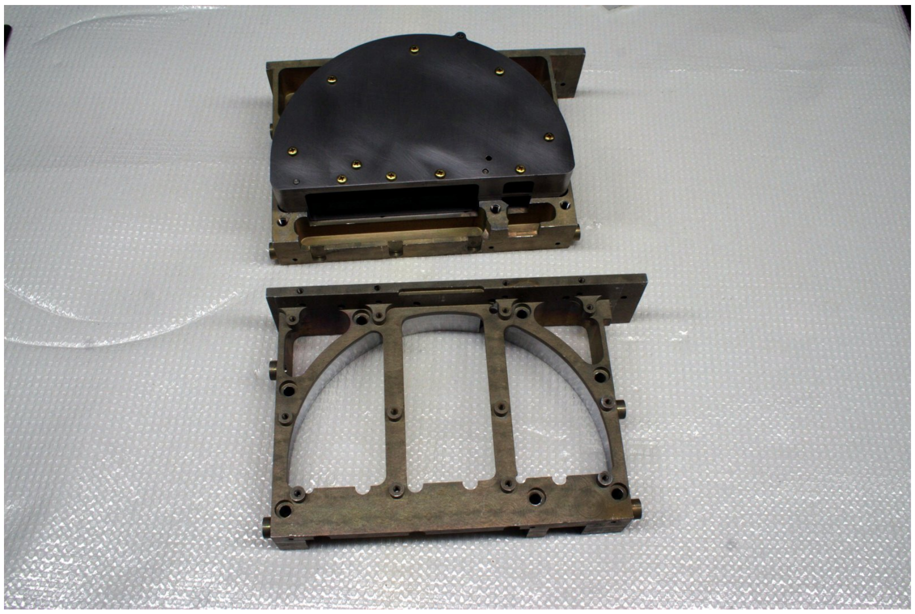


Figure C.9: The magnetic circuit is placed inside one of the clamshell halves.

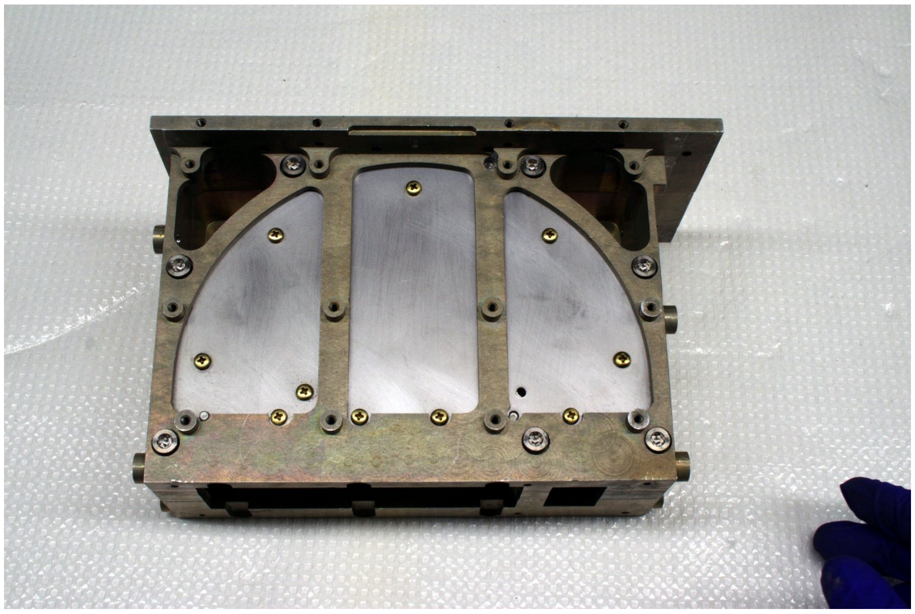


Figure C.10: The other half of the clamshell is attached and bolted to secure the magnetic circuit inside. The magnetic circuit itself is not fastened, but held securely by the close fitting clamshell.

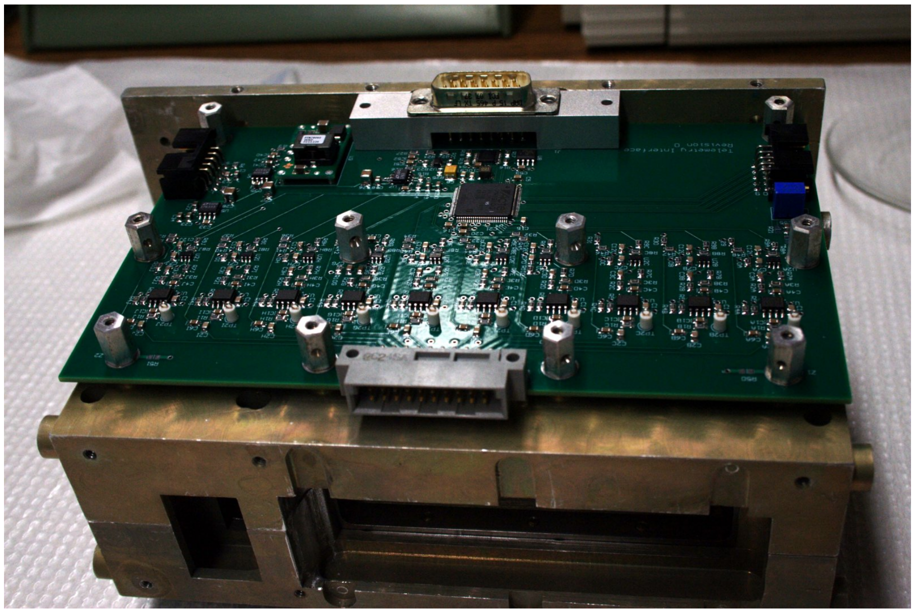


Figure C.11: The two telemetry/amplifier boards are fastened to each side of the clamshell with standoffs.

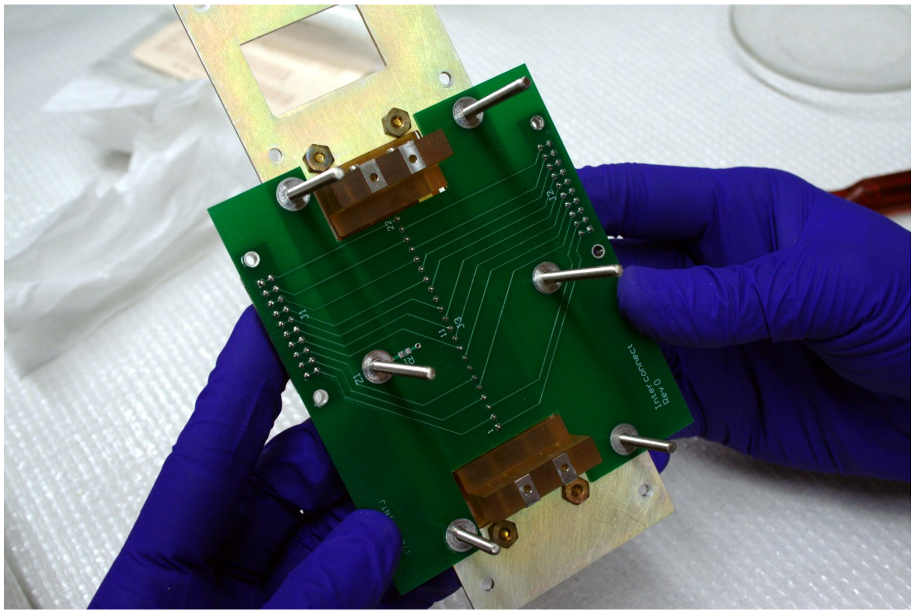


Figure C.12: The distribution board is fastened to the MCP stack. Spacers are threaded on to secure the board.

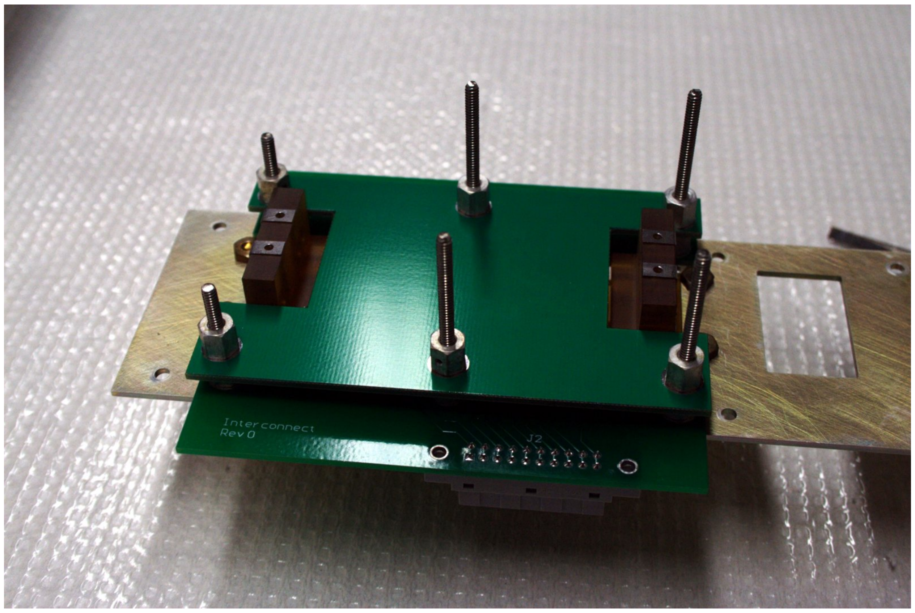


Figure C.13: The ground plane board is attached with another set of threaded spacers.

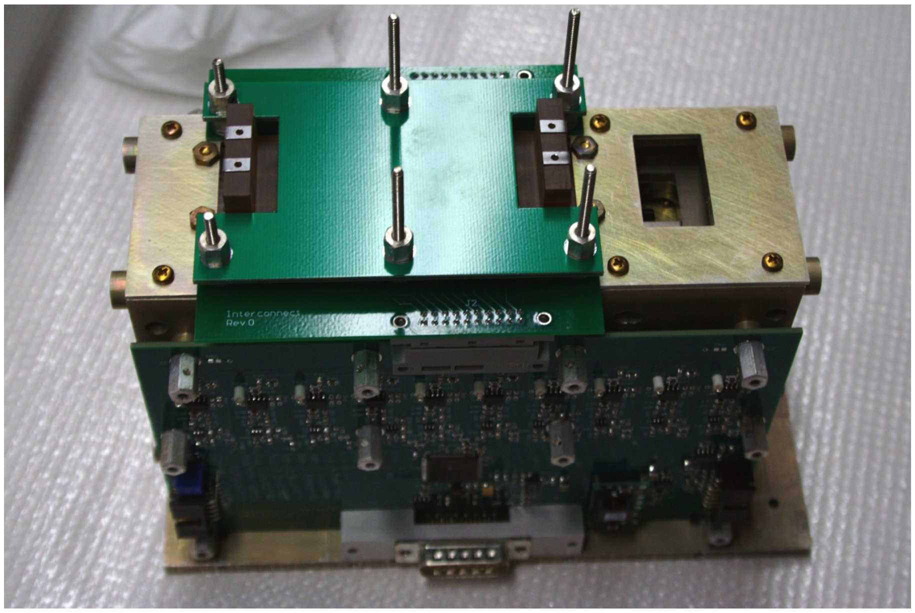


Figure C.14: The MCP stack is inserted into the magnetic circuit, and the distribution board connects to each telemetry/amplifier board.

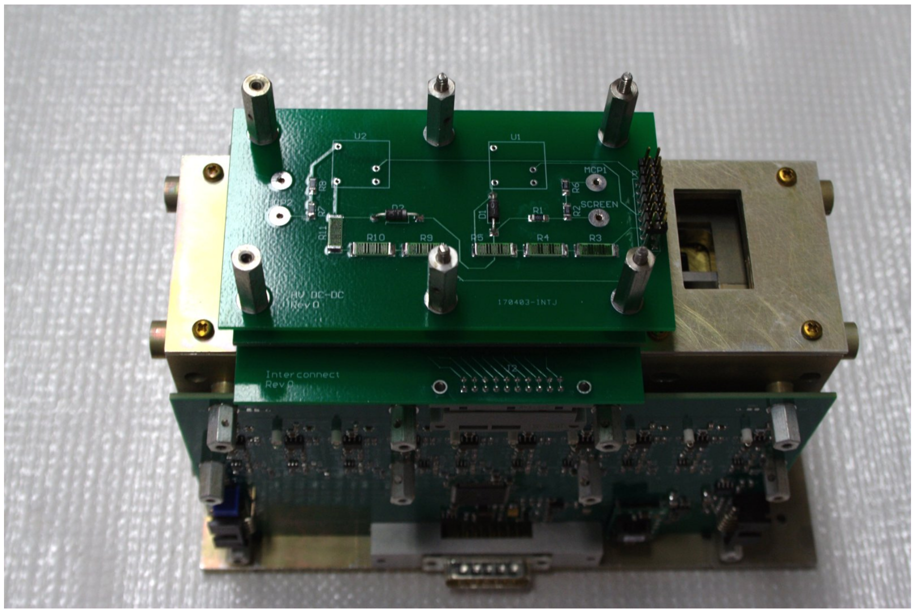


Figure C.15: The high voltage board is connected to the electrical stack assembly. Note that the high voltage supplies are not connected yet in this photo.

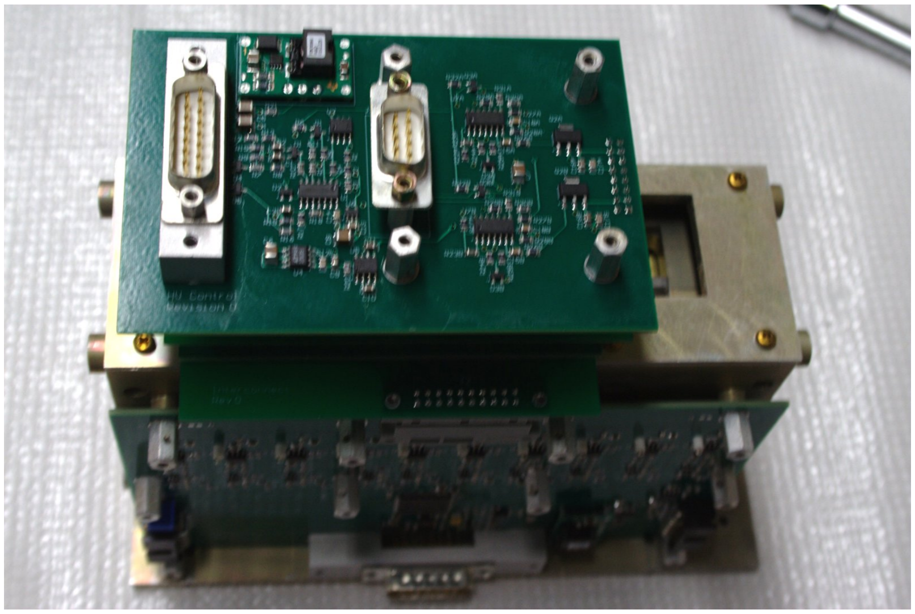


Figure C.16: The high voltage control board is the final component of the electrical stack assembly.

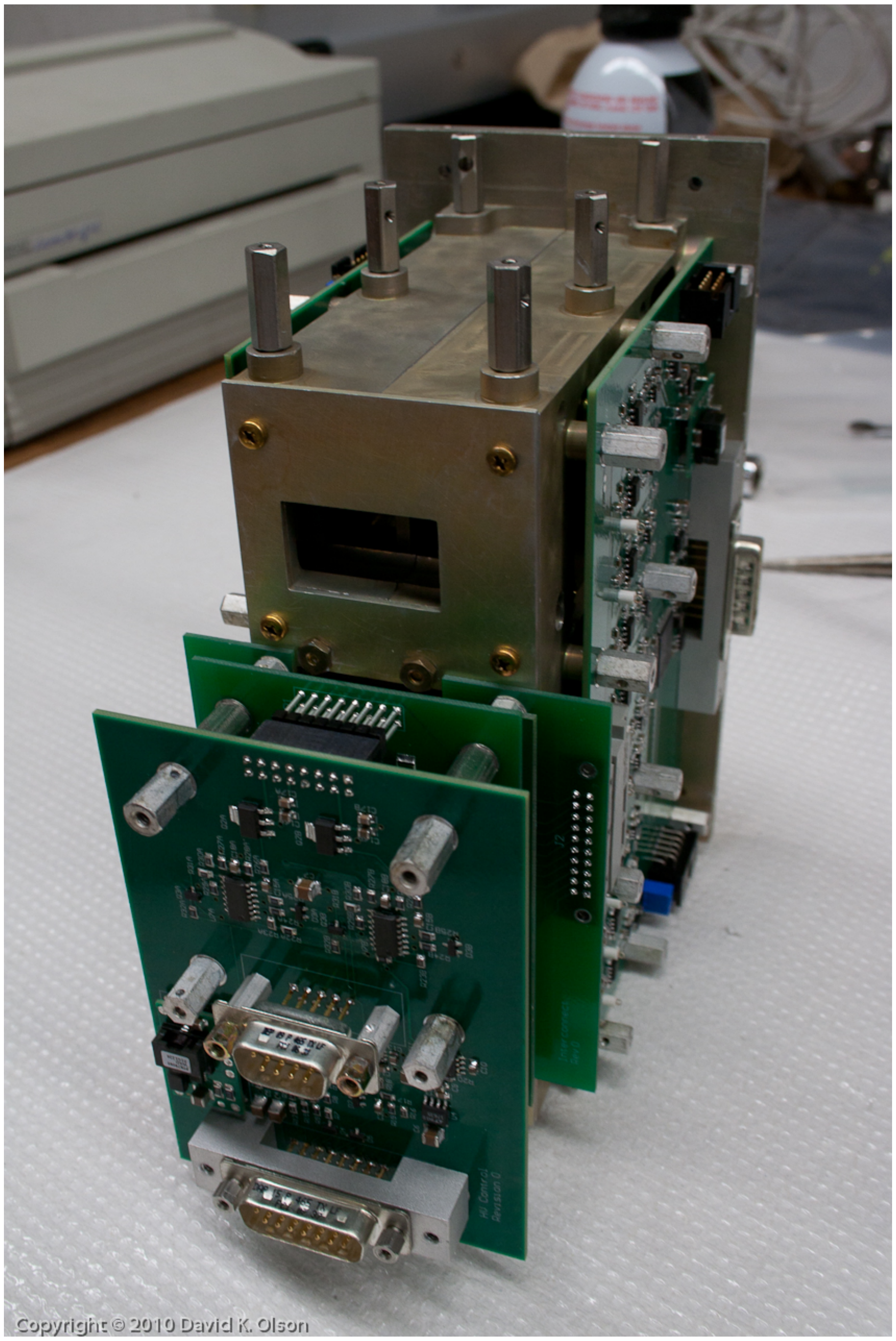


Figure C.17: Spacers are placed on the top of the clamshell.

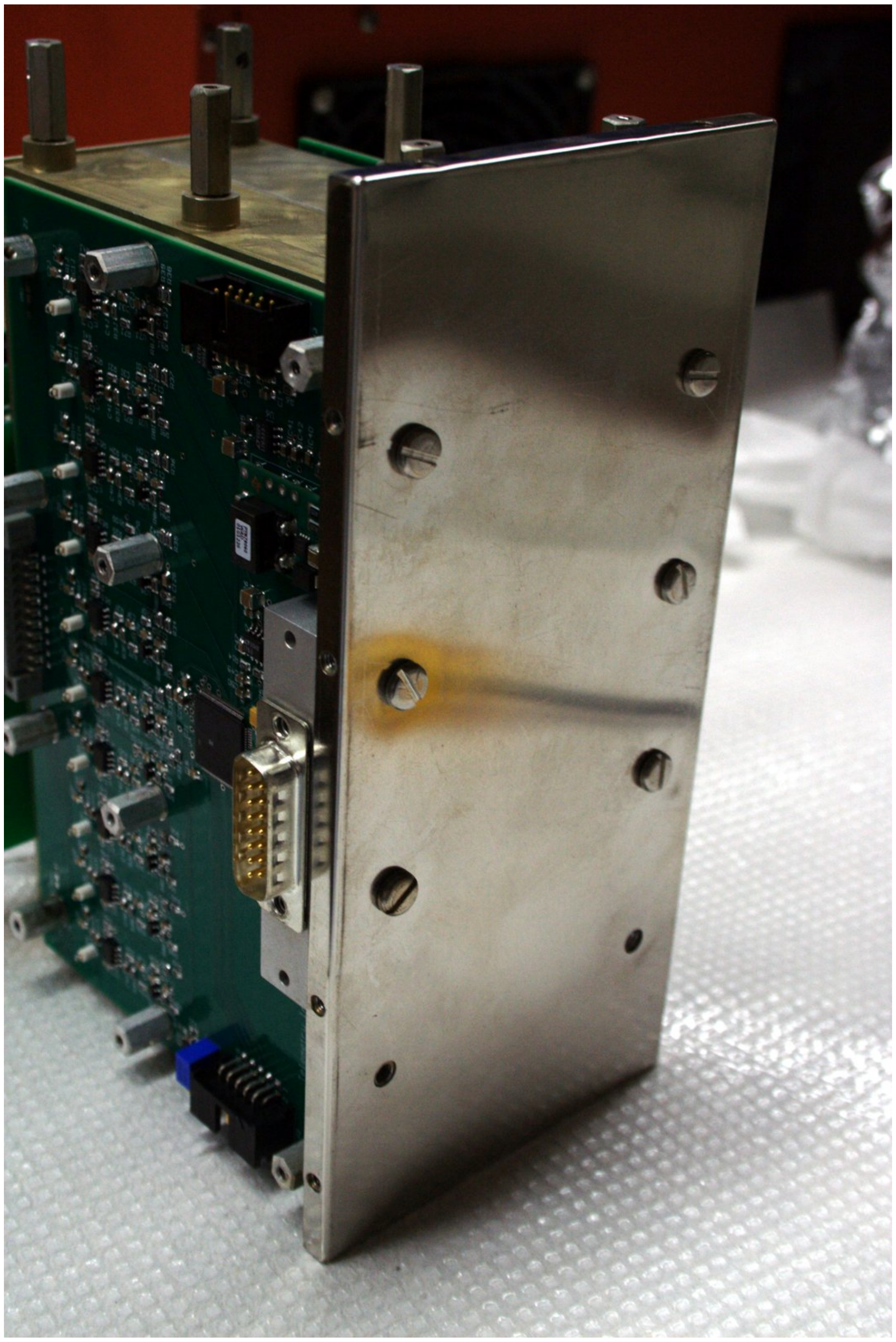


Figure C.18: The tray of the KIMS cover is fastened to the back plate.

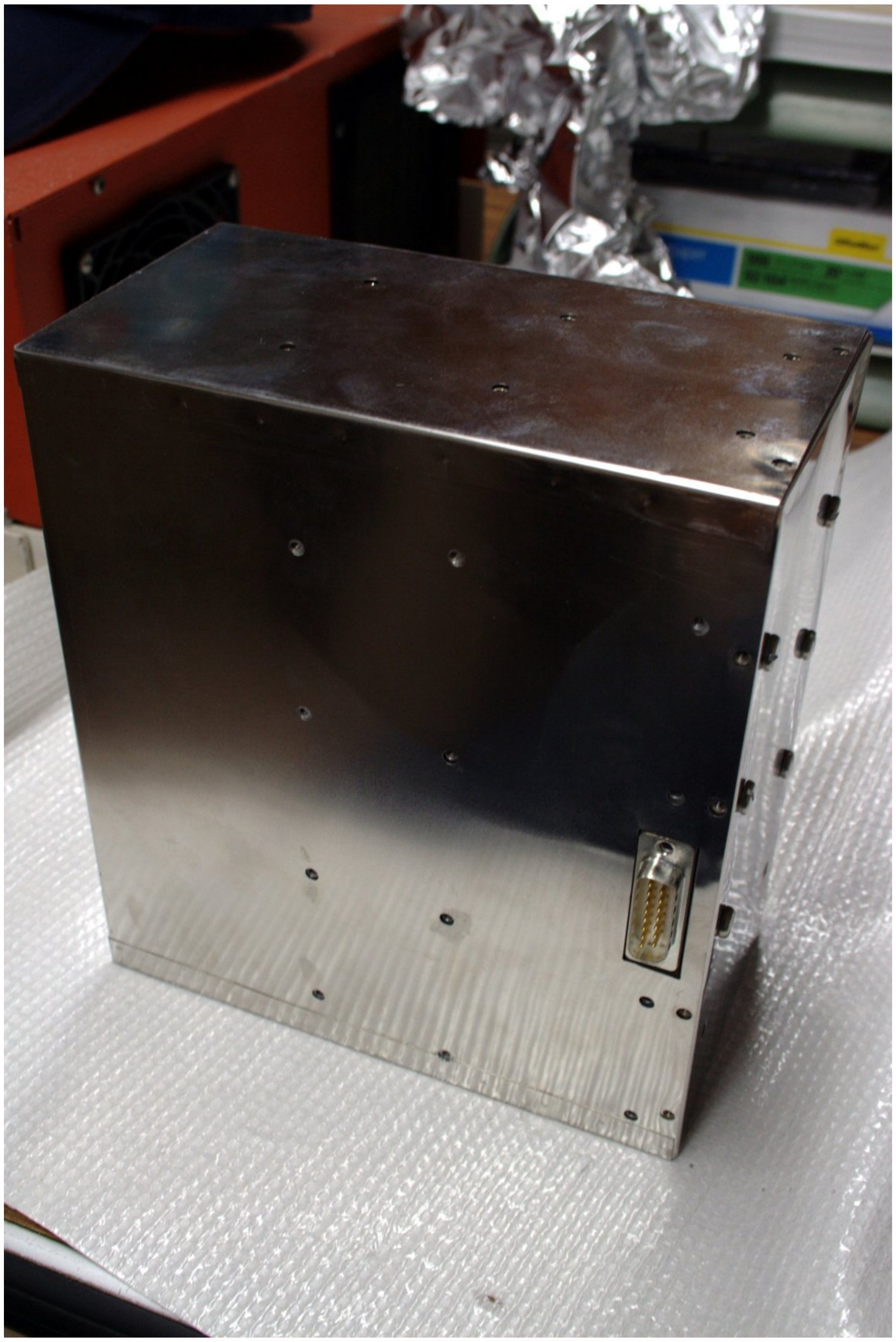


Figure C.19: The cover is slid onto the instrument. The material has just enough give to slide over the DB-15 connections on the telemetry/amplifier boards.

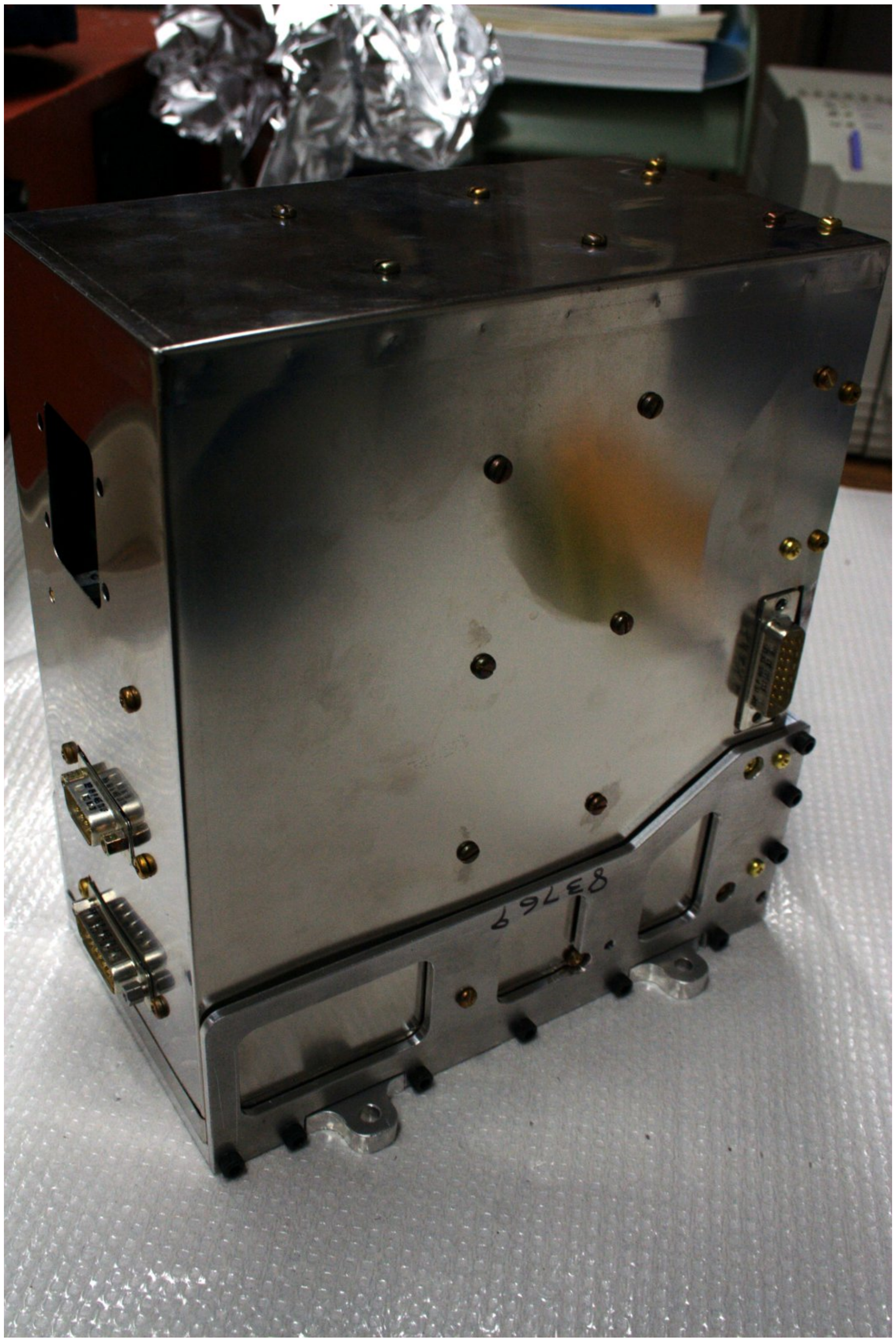


Figure C.20: The KIMS instrument completely assembled. The cover is fastened to the standoffs on the clamshell to prevent vibration at launch. The entire instrument is attached to an aluminum bracket for mounting on the lower bulkhead of the RENU payload.

Testing Photos

C.3 Vibration Testing

The following pages show the vibration test results for the KIMS instrument. The X, Y, and Z axes are as denoted in Fig. C.21. Natural modes are seen in all three directions at frequencies of 100's of Hz, but at the vibration levels of launch are not of concern. Note that the Z-axis is the thrust axis of the payload.

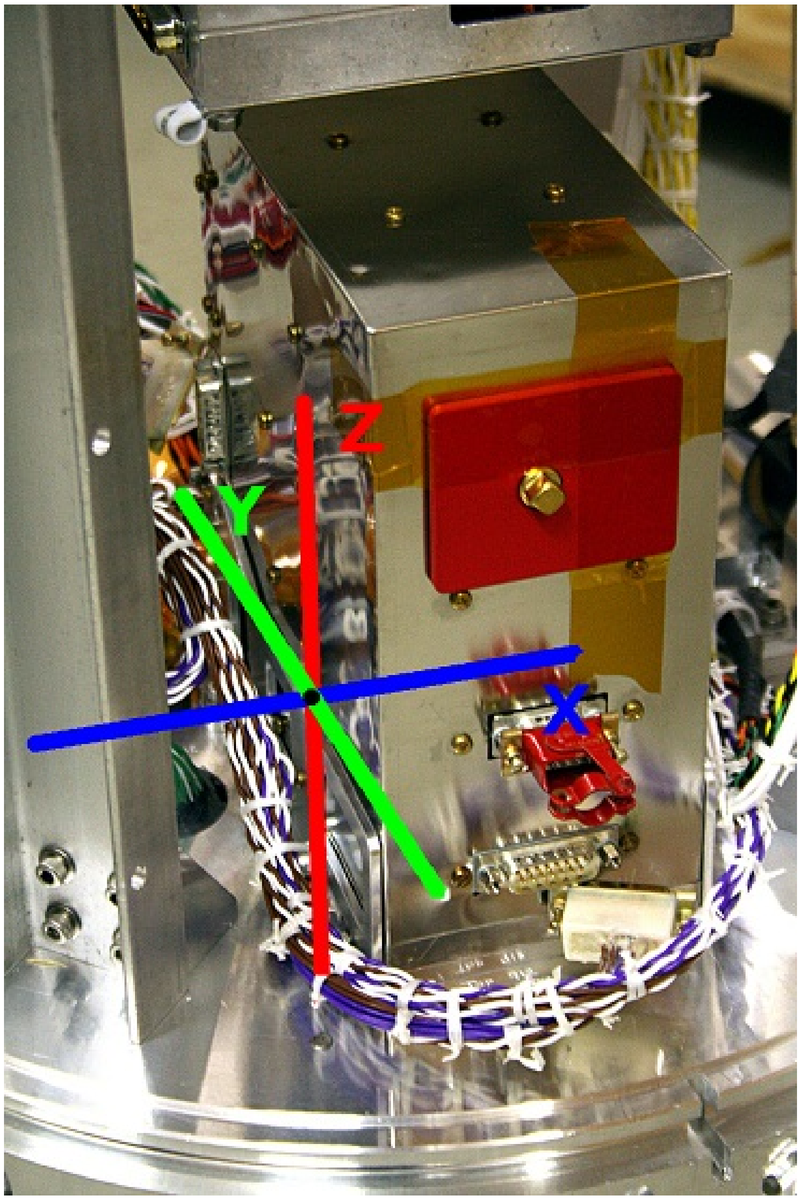
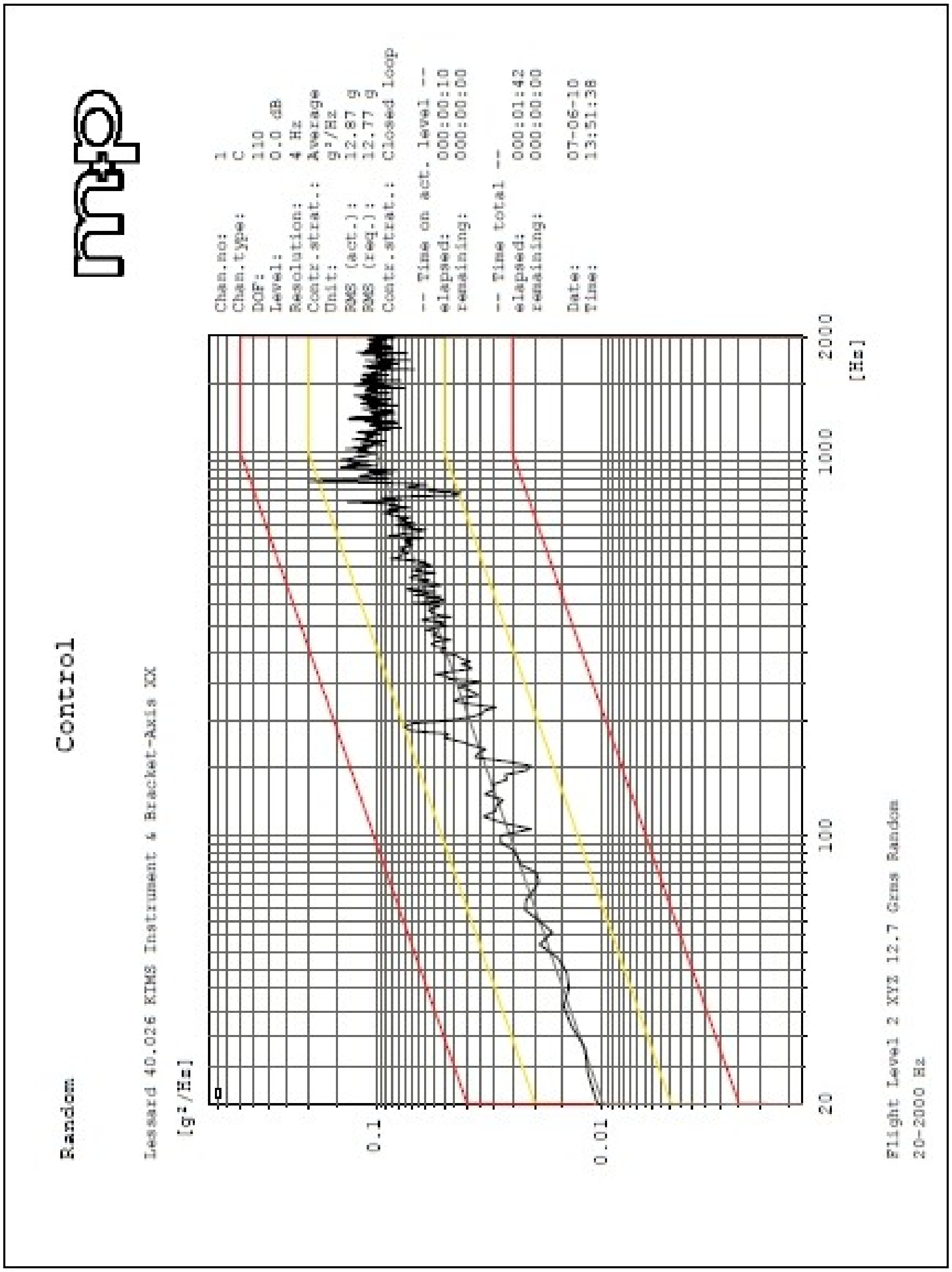
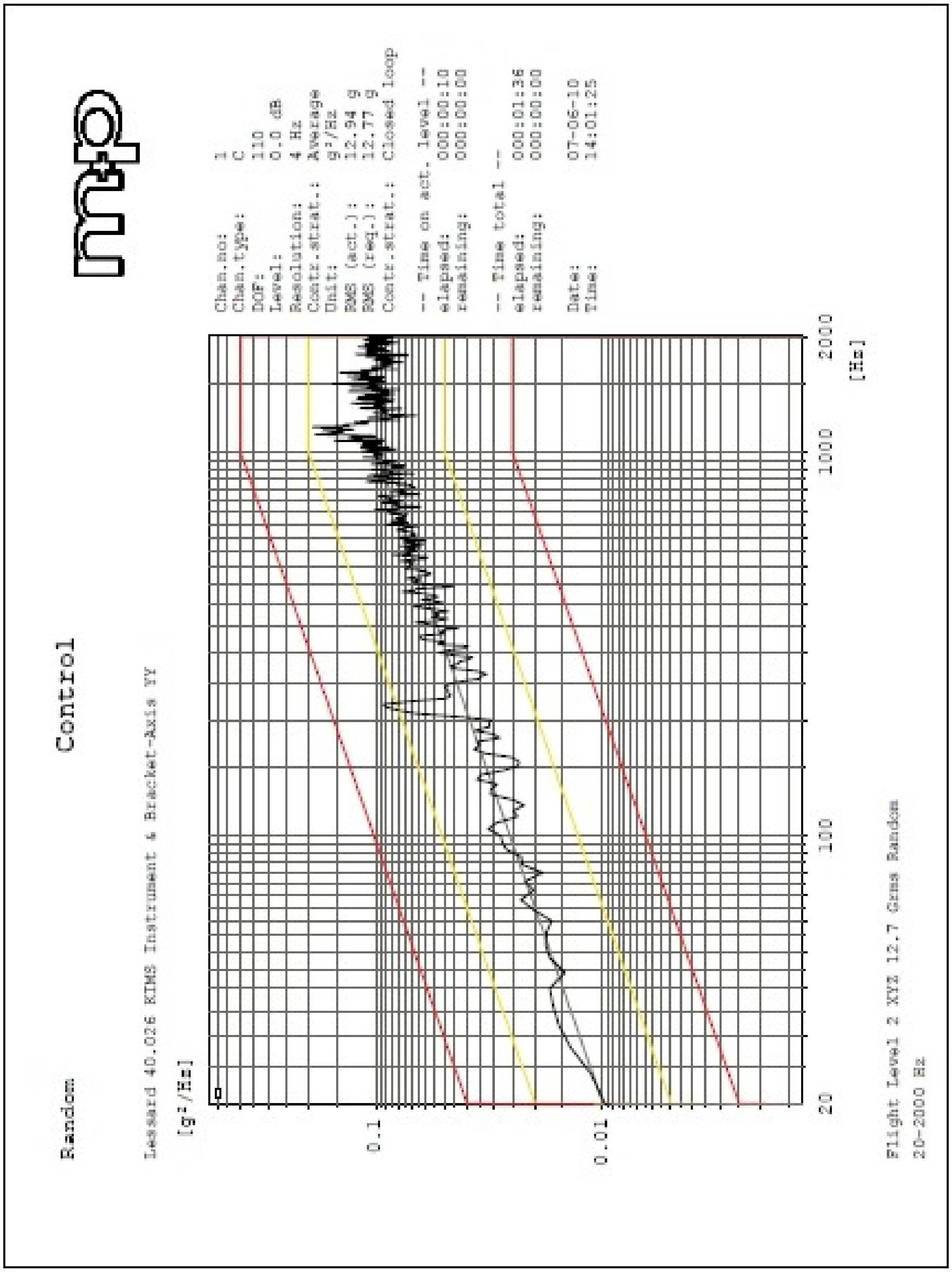


Figure C.21: Axis definitions for vibration testing.



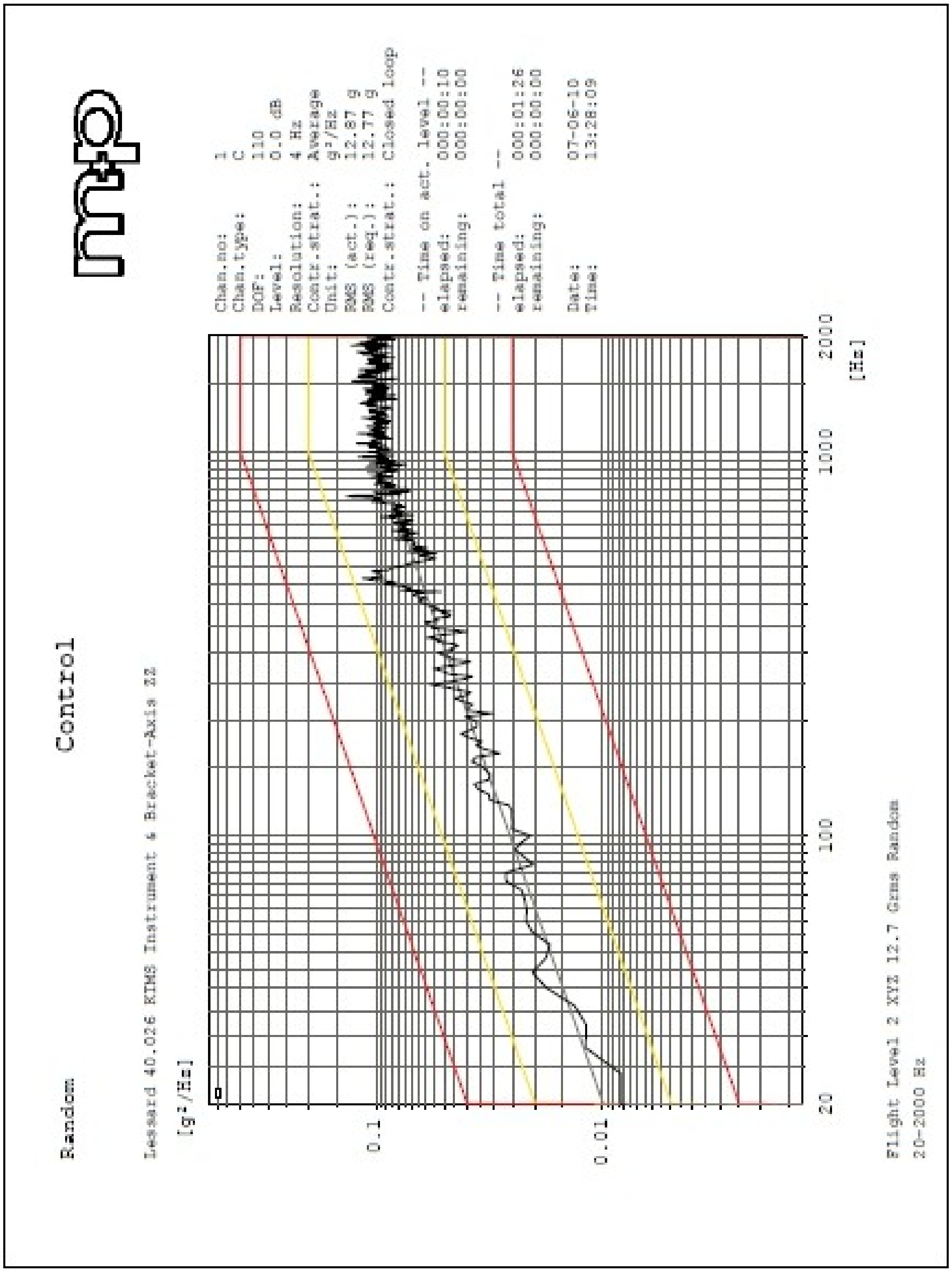
```
Chan.no:        1
Chan.type:      C
DOF:            110
Level:          0.0 dB
Resolution:     4 Hz
Contr.strat.:   Average
Unit:           g²/Hz
RMS (act.):     12.87 g
RMS (req.):     12.77 g
Contr.strat.:   Closed loop

-- Time on act. level --
elapsed:        000:00:10
remaining:      000:00:00

-- Time total --
elapsed:        000:01:42
remaining:      000:00:00

Date:           07-06-10
Time:           13:51:38
```

Figure C.22: White noise vibration test result of the KIMS instrument X-axis.



```
Random                 Control

Lessard 40.026 KIMS Instrument & Bracket-Axis YY

Chan.no:       1
Chan.type:     C
DOF:           110
Level:         0.0 dB
Resolution:    4 Hz
Contr.strat.:  Average
Unit:          g²/Hz
RMS (act.):    12.94 g
RMS (req.):    12.77 g
Contr.strat.:  Closed loop

-- Time on act. level --
elapsed:       000:00:10
remaining:     000:00:00

-- Time total --
elapsed:       000:01:36
remaining:     000:00:00

Date:          07-06-10
Time:          14:01:25

Flight Level 2 XYZ 12.7 Grms Random
20-2000 Hz
```

Figure C.23: White noise vibration test result of the KIMS instrument Y-axis.



```
Chan.no:        1
Chan.type:      C
DOF:            110
Level:          0.0 dB
Resolution:     4 Hz
Contr.strat.:   Average
Unit:           g²/Hz
RMS (act.):     12.87 g
RMS (req.):     12.77 g
Contr.strat.:   Closed loop

-- Time on act. level --
elapsed:        000:00:10
remaining:      000:00:00

-- Time total --
elapsed:        000:01:26
remaining:      000:00:00

Date:           07-06-10
Time:           13:28:09
```

Figure C.24: White noise vibration test result of the KIMS instrument Z-axis.

C.4 Magnetic Testing

The following pages show the configuration of the magnetometers for measuring the external fields of the KIMS instrument.

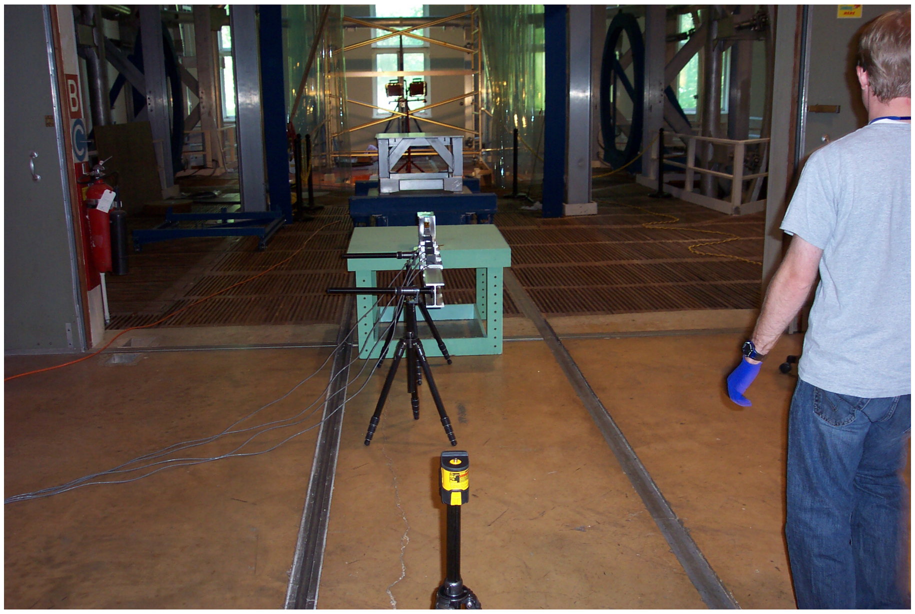


Figure C.25: Configuration for magnetic calibration

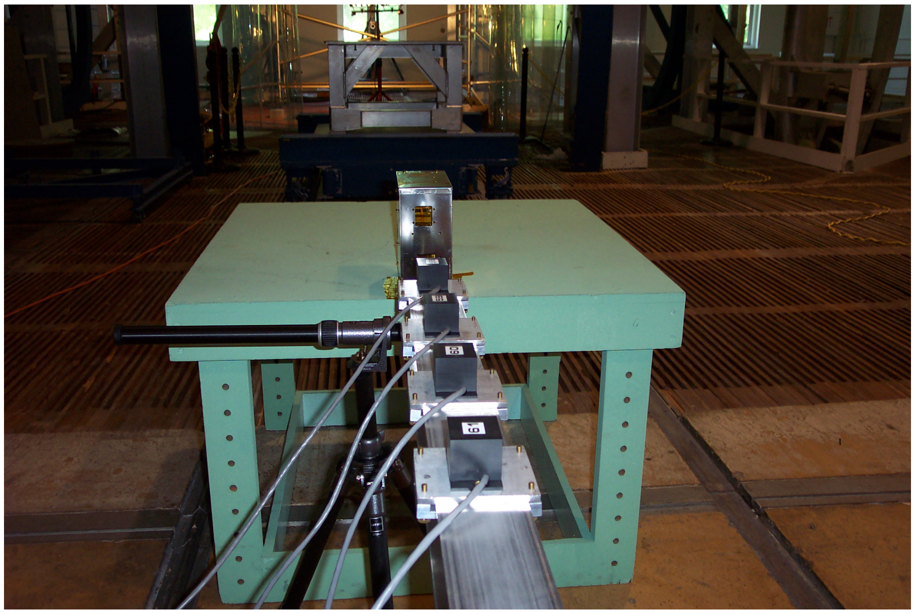


Figure C.26: Close up view of the magnetometer spacing at calibration

C.5 Integration Photos

The following pages include photographs from the final integration of the KIMS instrument on the RENU payload in Andøya, Norway.

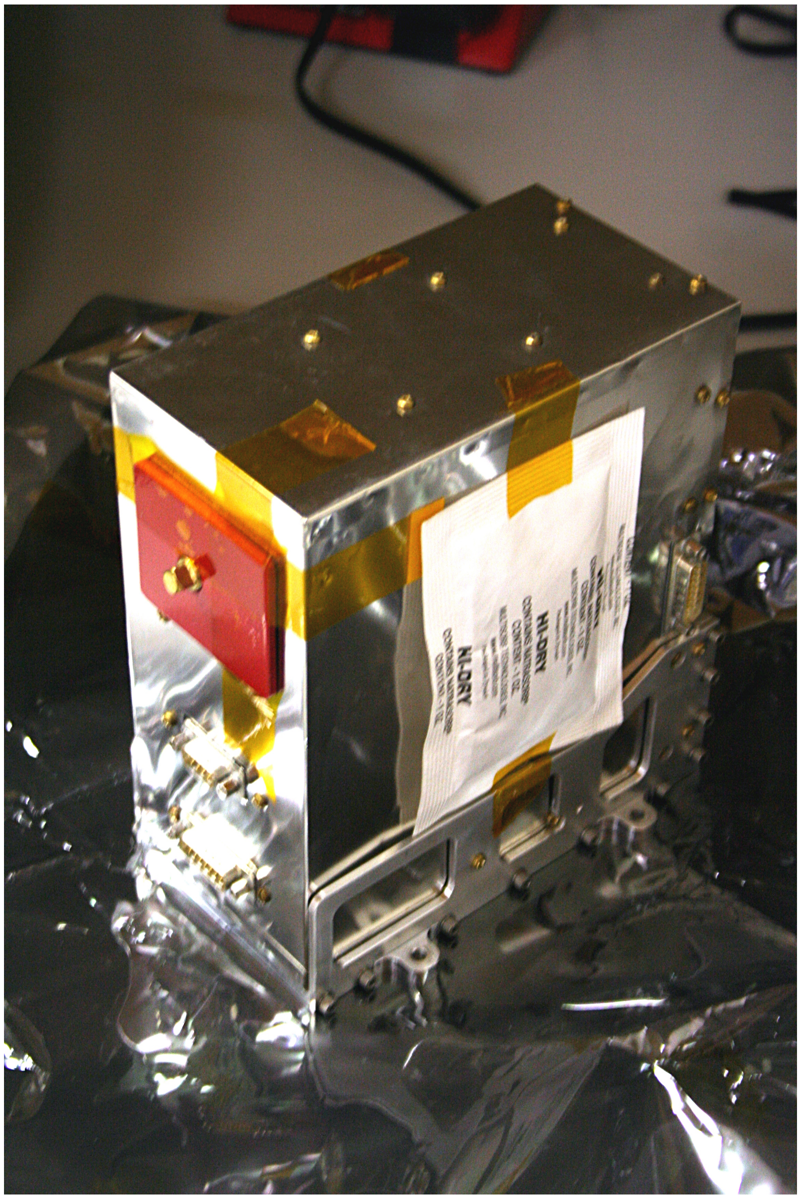


Figure C.27: Final state of the KIMS instrument prior to integration with the RENU payload.

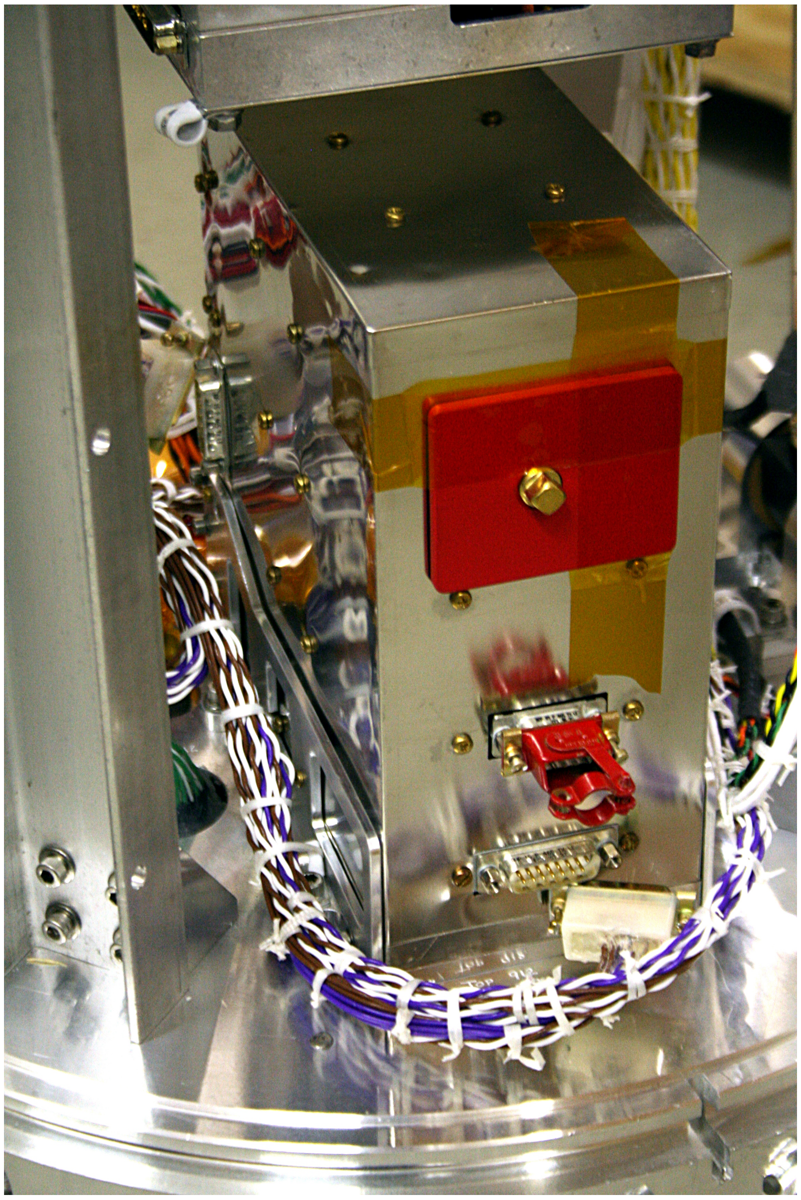


Figure C.28: KIMS mounted at the base of the RENU payload

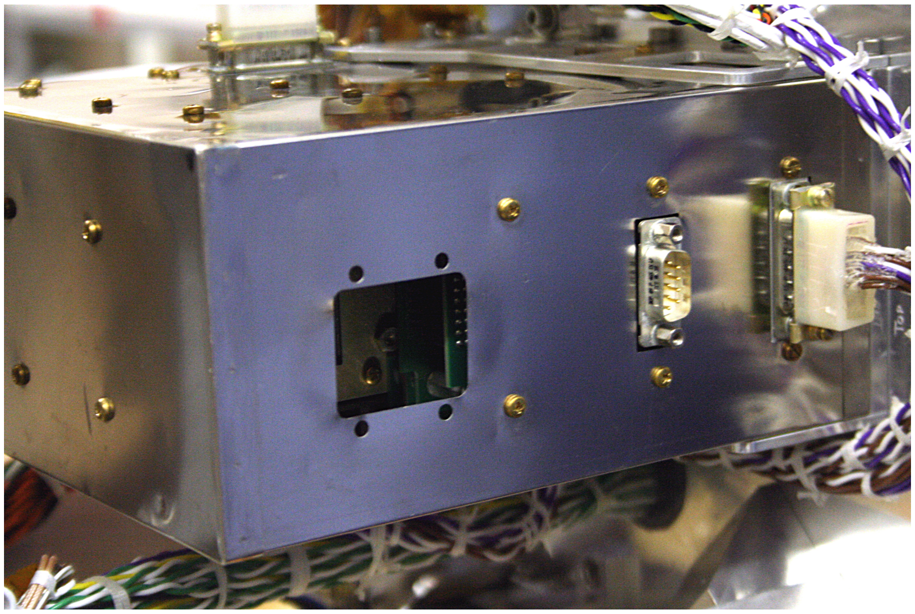


Figure C.29: KIMS with aperture exposed and high-voltage safety removed, ready for launch

Appendix D

RENU Launch Information

D.1 General Overview of Sounding Rocket Campaign RENU

RENU is a sounding rocket project for investigating neutral upwelling in the cusp. The rocket launched in the winter of 2011 from Andøya Rocket Range. RENU is designed to transit the cusp region during a neutral upwelling event, equipped with a suite of instruments that will build on previous observations of this phenomenon, as well as acquire new types of data to provide a fresh perspective on this problem. Successful acquisition of these data provide fundamental information, essential for the advancement of our understanding of this problem. The specific objectives of the mission are:

1. To measure netural gas, ion and electron temperature enhancements, which will provide an initial assessment of the upwelling process.

2. To measure large– and small–scale Joule heating in the cusp during the RENU flight. Large–scale data will be acquired by EISCAT; small–scale data (perhaps associated with Alfvén waves) will be acquired using onboard electric field measurements.

3. To measure the precipitating electron energy input.

4. To use measured quantities as inputs to thermodynamic and electrodynamic models for comparison to the observed upwelling.

D.1.1 Investigators and Team

Principle Investigator	Dr. Marc Lessard, University of New Hampshire
Co-investigators	Dr. Jim Clemmons, Aerospace
	Dr. James Hecht, Aerospace
	Dr. Payl M. Kintner, Cornell University
	Dr. Kristina Lynch, Dartmouth College
	Dr. Matthew G. McHarg, USAF Academy
Engineering Designers	Dr. Parris Neal, USAF Academy
	Dr. Kevin G. Rhoads, Dartmouth College
	David K. Olson, University of Maryland
	Steven Powell, Cornell University
	Paul Riley, University of New Hampshire
Mission Manager	Jay Scott, NSROC
Project Manager	Libby West, NASA Goddard Space Flight Center
Flight Performance	Mike Disbrow, NSROC
Mechanical Systems	Shane Thompson, NSROC
Mechanical Technician	Clay Merscham, NSROC
Telemetry Systems	Jim Diehl, NSROC
Power Systems	Tom Malaby, NSROC

Attitude Control System (ACS)	Valerie Gsell, NSROC
Vehicle Systems	Nick Wroblewski, NSROC
SQA	James Alexander, NSROC
Ground Safety	Chico Ayers, CSC
Flight Safety	Jim Veney, NASA Goddard Space Flight Center

D.2 Vehicle Requirements and Performance

D.2.1 Trajectory Data/Attitude Solution

Absolute trajectory knowledge is required at the 500 meter level. These requirements are satisfied by the accuracy of the data supplied by the NSROC and Cornell GPS receivers. The science team requires an attitude solution with an accuracy within one degree.

D.2.2 Outgassing, Magnetic Sensitivity, RFI Susceptibility

The Dartmouth/UNH particle detector experiments on the main payload are sensitive to payload outassing and steps need to be taken to keep outgassing to a minimum. These steps include proper material selection (see NASA Reference Publication 1124) and payload cleanliness.

D.2.3 Coning Angle

Prior to subpayload ejection, the main payload coning should be driven as close as possible to zero and the payload aligned to the background magnetic field. In order to keep the payload within the deadband, ACS updates may be required, although we emphasize that particle data acquired during ACS maneuvers are not usable and, so, ACS maneuvers should be kept to a minimum by ensuring that the payload is very well balanced.

The final spin rate of the payload should be 0.5 Hz. While the accuracy of the actual final spin rate is not critical, the imagers will need to 'despin' and will need an accurate measure of the roll rate provided continuously.

D.2.4 Despin

After all deployments, a final main payload spin rate of approximately 0.5 Hz is desired to ensure payload stability and achieve the scientific objectives.

D.2.5 Horizon Sensors/Sun Sensors

A horizon sensor and/or a sun sensor will be needed for each payload to assist in the attitude solution.

D.2.6 Range Support

- Dry nitrogen purge of payload (particle detectors, imager) required during build–up and on launcher. This minimizes the moisture absorption of the particle detectors. Once the nose cone is installed, the dry nitrogen purge is to be connected to the payload skin through a “fly–a–way” disconnect.

- Liquid Nitrogen cooling support for Aerospace PMT
- Standard Wallops “lunchbox” parallel interface for real time UNH imager system
- Distribution of real time trajectory data and/or look angles for use by the ground based imaging instruments at the Longyearbyen science station
- On-site generation of flight telemetry data on CD within 4 hours of launch using Programmable Telemetry Processor (PTP) with data in PTP Stamp time format.
- TDP bit sync for Aerospace Instrumentation, both in the TM building and at the science center

D.2.7 Launch Conditions

- Andøya Rocket Range — Nov 28 to Dec 12
- 5 hour period within 0300–1100 UT, nominally 0600–1100 UT (0700–1200 Local Time)
- Moon in last or first quarter or below the horizon at Longyearbyen
- Solar depression angle greater than 10 degrees at Longyearbyen

- The payload must be in sunlight to permit nominal functioning of the on-board solar aspect sensor during the prime data taking period (above 500 km altitude)

- Azimuth and launch angle chosen for apogee over Svalbard at 1000 km down-range — this is northward. We desire the apogee point to be chosen such that the projection of the B–field vector at apogee down to 100 km altitude results in a point directly overhead Longyearbyen. If this is not possible, then the magnetic footprint of the trajectory should pass overhead Longyearbyen.

- It will be necessary to hold the count at T minus 2 minutes for up to 30 minutes at a time

Launch requires:

- An active cleft ionosphere with 5577 and 6300 light emissions overhead at either Longyearbyen of Ny–Ålesund.

- At least 20 nT of magnetometer activity at either Longyearbyen or Ny–Ålesund.

- Clear skies at either Longyearbyen or Ny-Ålesund such that either auroral TV or meridian scanning photometers provide data.

- EISCAT Svalbard Radar recording ion outflows and elevated electron temperaturs.

- No fishing vessels in impact area.

D.3 Success Criteria

D.3.1 Comprehensive Mission Success

Comprehensive success means that we meet the minimum success criteria and that we also acquire supportive data, including in–situ observations of:

1. Apogee of 500 km
2. Onboard auroral images
3. Upward looking PMTs
4. Medium energy ions (to observe outflow)
5. Ion composition (BEEPS)
6. Energetic ions (KIMS)

D.3.2 Minimum Success

Minimum success means that the data needed to provide a basic assessment of thermospheric upwelling associated with ionospheric processes is acquired. This requires the following in-situ measurements, acquired in the vicinity of the EISCAT radar:

1. Apogee of at least 410 km
2. Electron temperature (ERPA)
3. Electron precipitation

4. Ion temperature
5. Neutral gas temperature
6. Electric fields
7. Magnetic fields

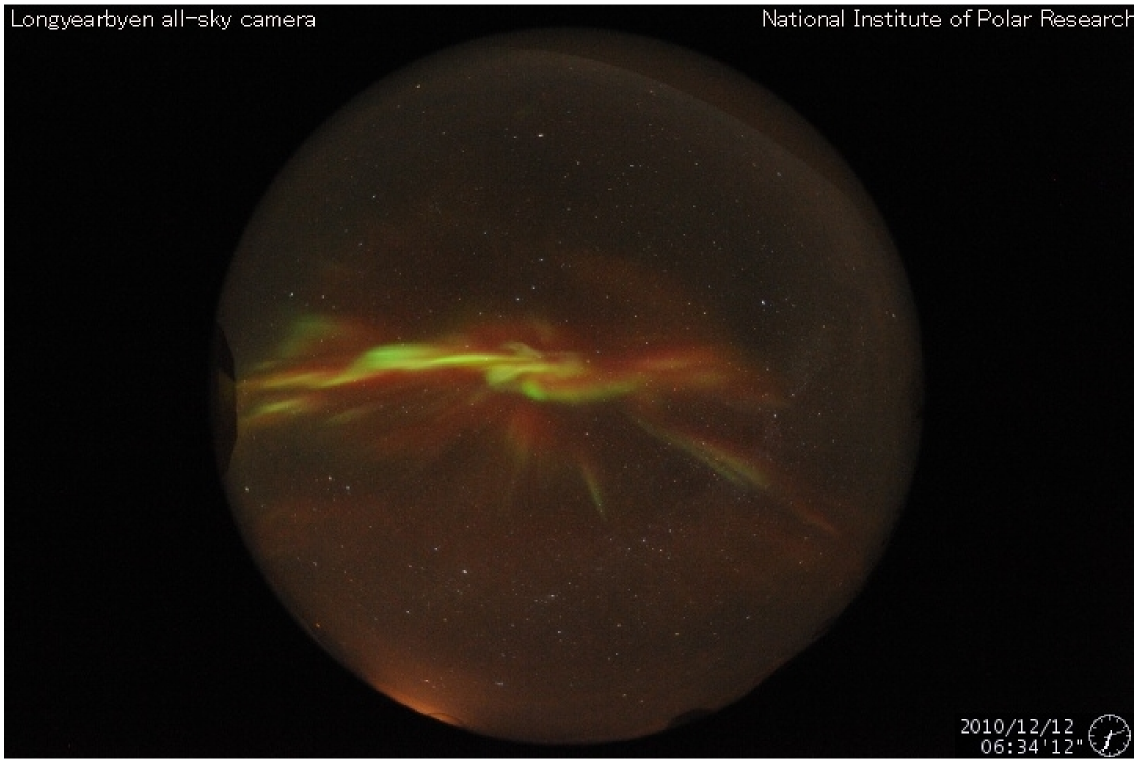
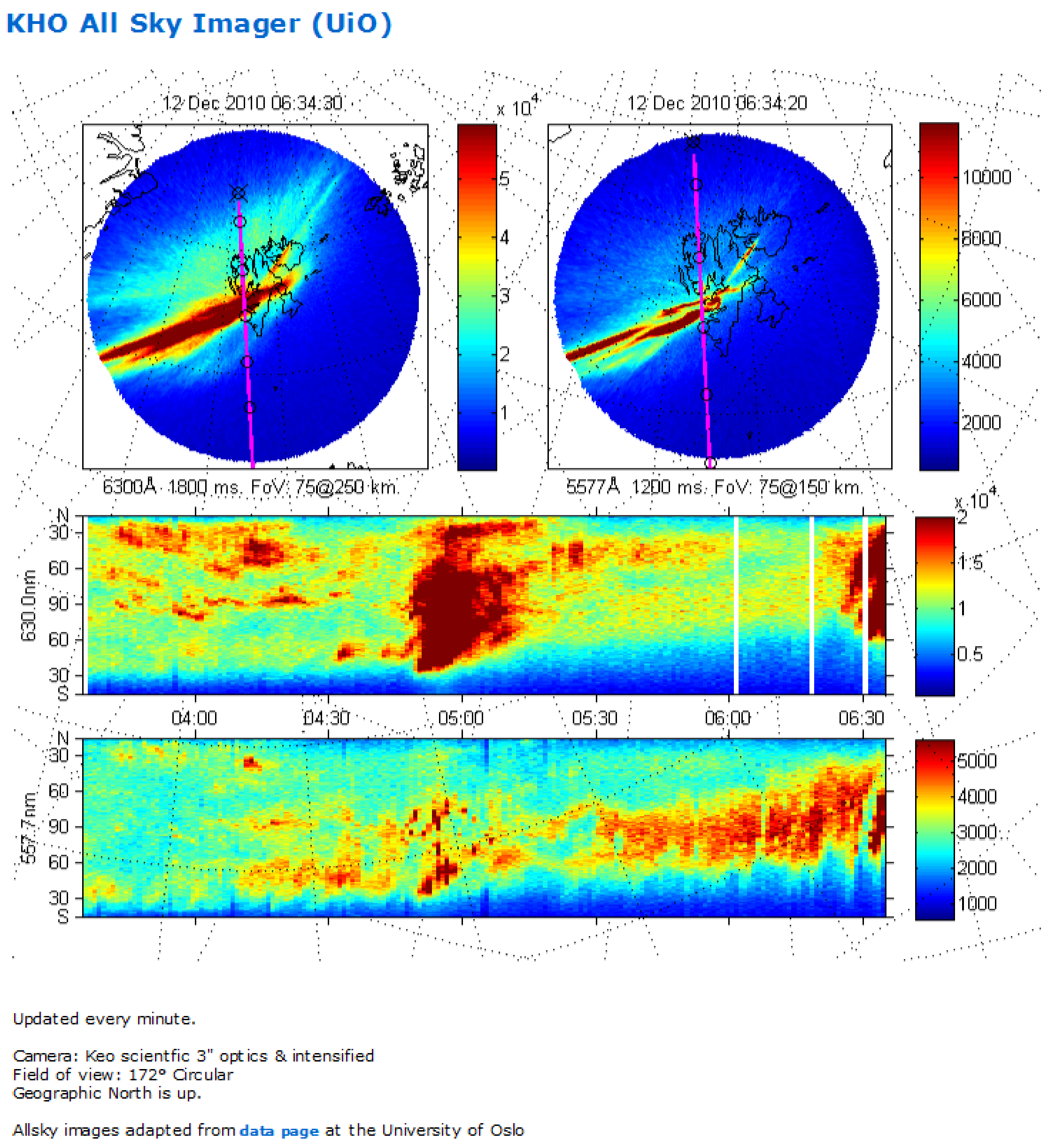

Longyearbyen all-sky camera
National Institute of Polar Research
2010/12/12
06:34'12"

Figure D.1: All-sky photograph of the aurora at launch.

D.4 RENU Launch Conditions

The RENU sounding rocket launched at 06:33 UT on 12 December 2010. The following images describe the conditions present when it was decided to launch.

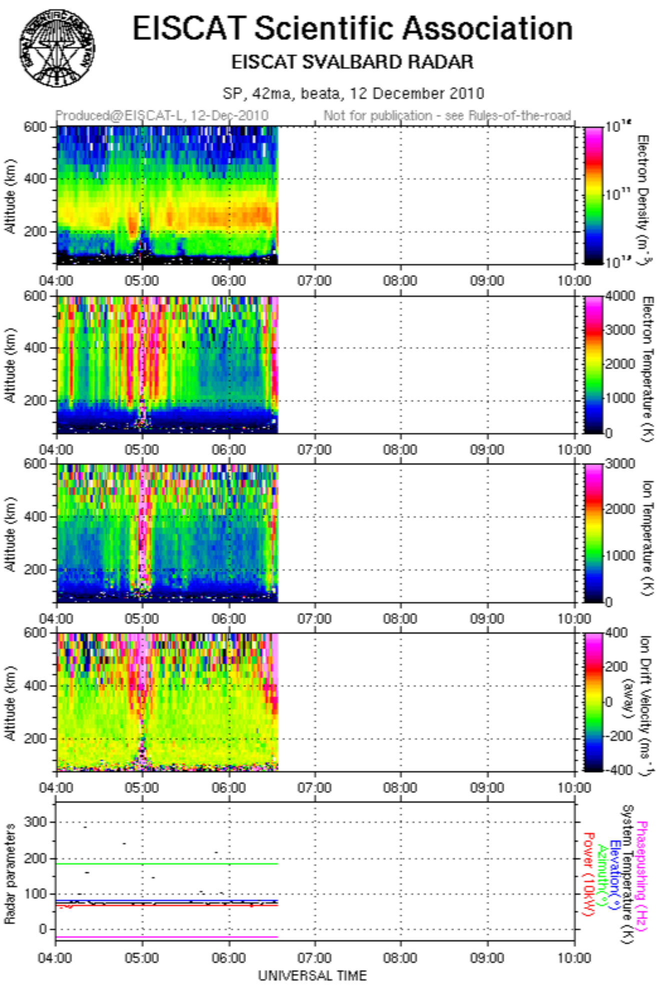


Updated every minute.

Camera: Keo scientfic 3" optics & intensified
Field of view: 172° Circular
Geographic North is up.

Allsky images adapted from data page at the University of Oslo

Figure D.2: All-sky view of red and green wavelengths from the observatory at Svalbard.



Auto-updating quarter-hourly [stop]

Figure D.3: EISCAT radar measurements of precipitating electrons.

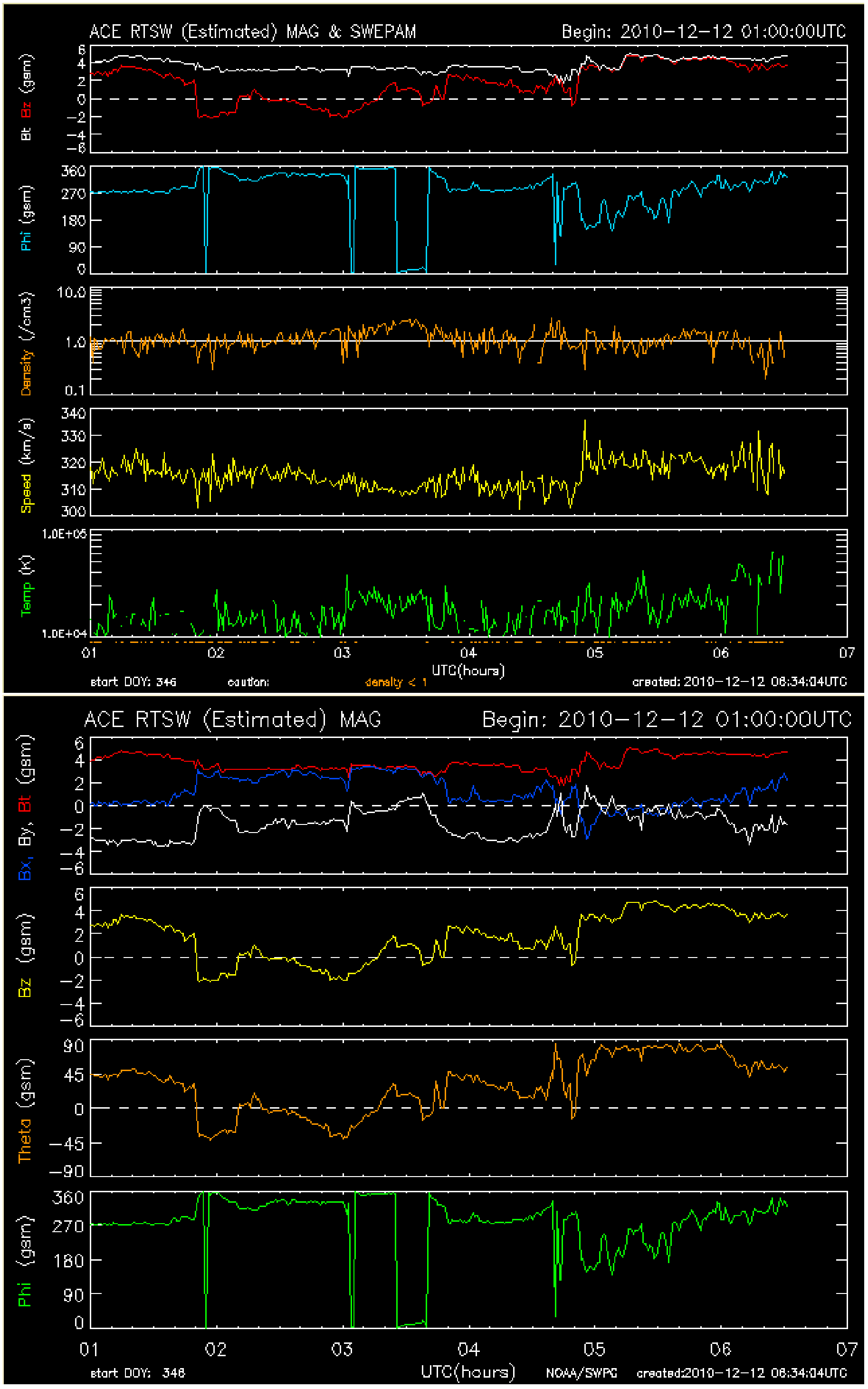


Figure D.4: Solar wind conditions measured by the ACE satellite.

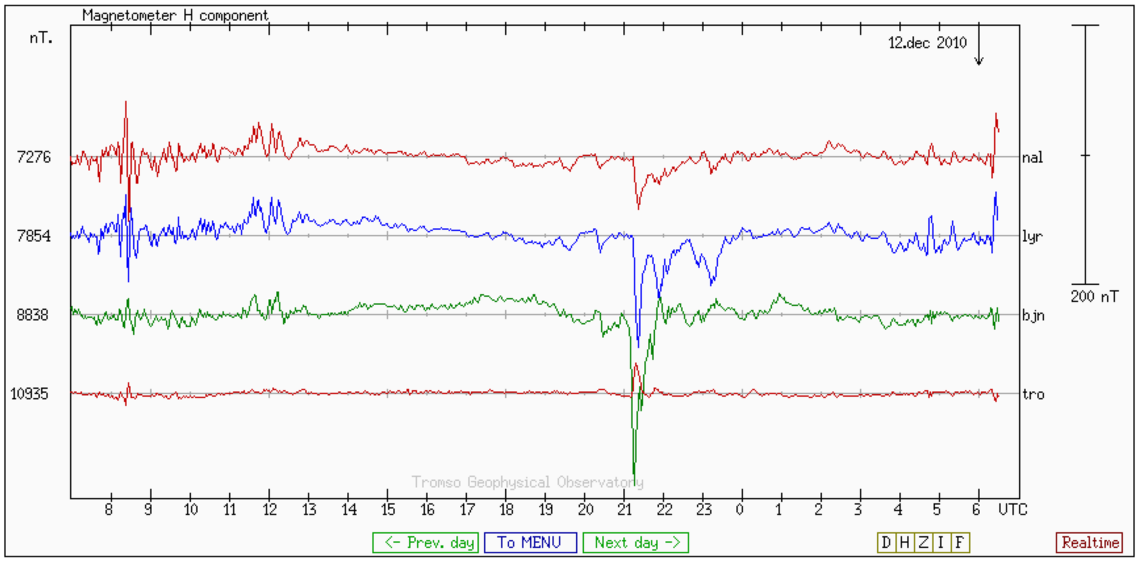


Figure D.5: Collected magnetometer measurements in the Svalbard and northern Norway regions.

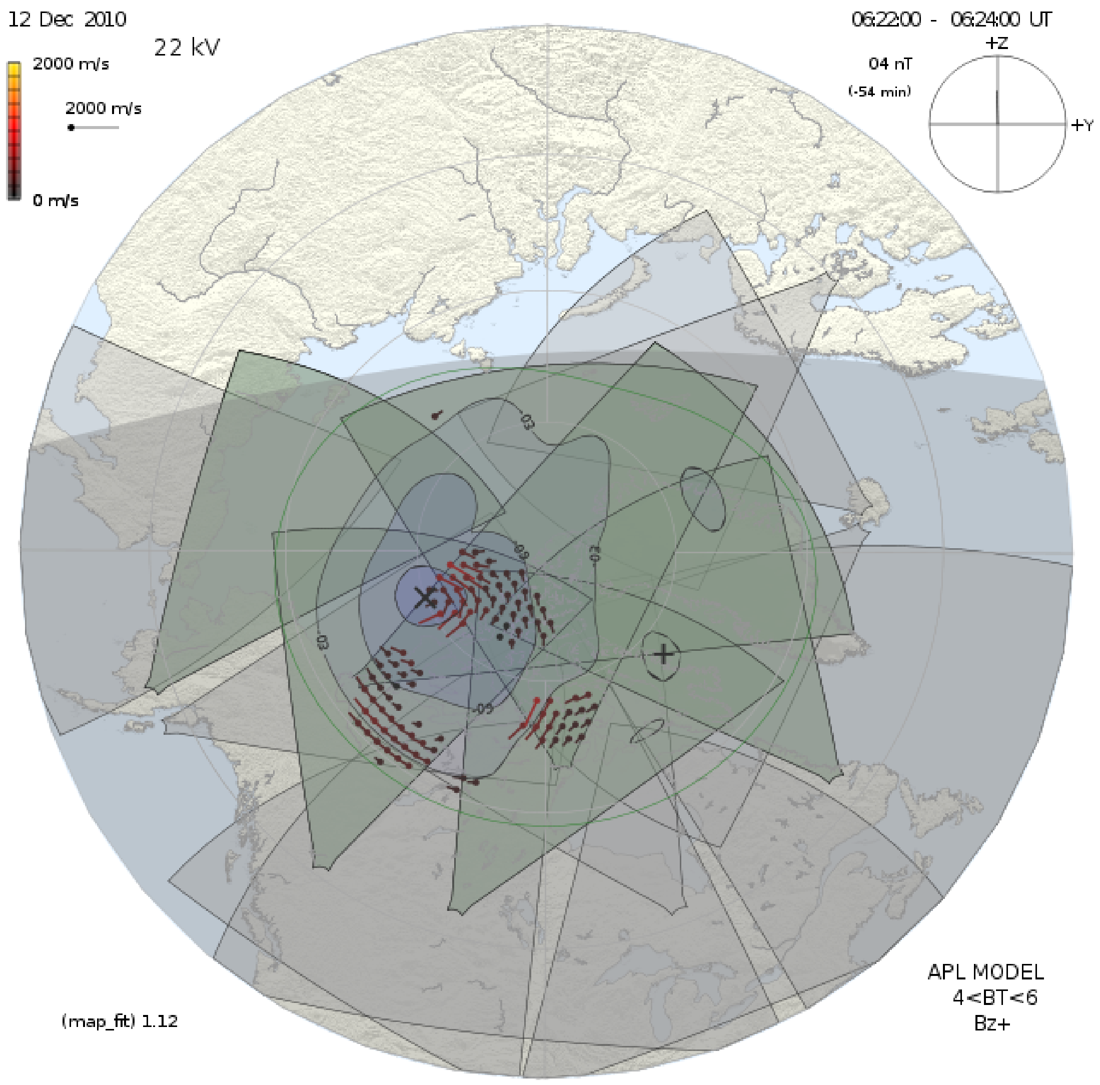


Figure D.6: SuperDARN radar measurements showing ionospheric convection.

Appendix E

Statistical Analysis Software

Some lines in this code are longer than will fit on a printed page. Places where an artificial line break has been inserted are marked with the characters ":::" at the beginning of the line.

E.1 FAST

E.1.1 FAST.py

```
# Routines for analysis of FAST data
# v1.7 changelog:
#   ::  changed to automatically deal with differing energy scales
#       for particle data
#   ::  1.6 error had ees energy > 4 and ies energy > 50. Corrected
#       to ees > 50 and ies > 4.

from sys import stdout,version
from nantools import nanmean,nanstd
from numpy import nan,isnan,array,zeros,arange,shape,sum,mean,log10,arccos
from pylab import find,contourf
from math import sin,cos,sqrt,pi
from scipy.interpolate import interp1d
from spacepy import pycdf
from time import time,gmtime
import pickle

softversion="FAST.py 1.71"
basedir="/home/david/research/FAST/data/"
f=open('FAST-values.pkl','rb')
standard_E=pickle.load(f)
f.close()
```

```
def chauvenet(data,window,weight):
    f_data=zeros(len(data))
    for i in range(len(data)):
        if i<window/2:
            m=nanmean(data[i:i+window/2])
            s=nanstd(data[i:i+window/2])
        elif (len(data)-i)<window/2:
            m=nanmean(data[i-window/2:])
            s=nanstd(data[i-window/2:])
        else:
            m=nanmean(data[i-window/2:i+window/2])
            s=nanstd(data[i-window/2:i+window/2])

        if abs(data[i]-m)>weight*s:
            f_data[i]=nan
        else:
            f_data[i]=data[i]

    return f_data

def smooth(data,r_data,interval):
    s_data=zeros(len(interval))
    for i in range(len(interval)):
        if i==0:
            r_min=interval[i]
        else:
            r_min=(interval[i]+interval[i-1])/2.
        if i==len(interval)-1:
            r_max=interval[i]
        else:
            r_max=(interval[i]+interval[i+1])/2.

        a=find(r_data<r_max)
        collect=find(r_data[a[0]:a[-1]]>r_min)
        s_data[i]=nanmean(data[collect])

    return s_data

def interpolate(x,y,x_new,width,int_kind='cubic'):
    y_new=zeros(len(x_new))
    for i in range(len(x_new)):
        j=0
        while(x[j]<x_new[i]):
            j+=1
        collect=arange(j,j+1)
```

```
            while x[collect[0]]>x_new[i]-width/2. and collect[0]!=0:
                collect=arange(collect[0]-1,collect[-1]+1)
            while x[collect[-1]]<x_new[i]+width/2. and collect[-1]!=len(x)-1:
                collect=arange(collect[0],max(collect)+2)
            if len(collect)<4:
                y_new[i]=nan
            else:
                f=interp1d(x[collect],y[collect],kind=int_kind)
                y_new[i]=f(x_new[i])

        return y_new

    def get_E(orbit,t_base,margin=30.,subset=False):
        dcefs=pycdf.CDF(basedir+orbit+'/dcefs_'+orbit+'.cdf')
        print("Getting Time array...")
        t_E=dcefs['TIME']

        if subset:
            ti=0
            tf=len(t_base)-1
            i=0
            while(t_E[i]<t_base[0]-margin):
                t_min=i
                i+=1
            print(" found t_min...")
            while(t_E[i]<t_base[-1]+margin):
                t_max=i
                i+=1
            print(" found t_max...")
        else:
            t_min=0
            t_max=len(t_E)
            i=0
            while(t_base[i]<t_E[0]):
                ti=i+1
                i+=1
            print(" found ti...")
            while(i<len(t_base) and (t_base[i]<t_E[-1])):
                tf=i
                i+=1
            print(" found tf...")
        tnew=zeros(t_max-t_min)
        for j in range(len(tnew)):
            tnew[j]=t_E[t_min+j]
        t_E=tnew
        print("  done!")

        print("Getting E_along V array...")
        E_V=zeros(len(t_E))
        for j in range(len(t_E)):
            E_V[j]=dcefs['E_ALONG_V'][t_min+j]
        print(" Filtering E_V...")
        E_V=chauvenet(E_V,100,2)
        t_avg=arange(t_E[0],t_E[-1]+4.,4.)
        print(" Smoothing E_V...")
        E_avg=smooth(E_V,t_E,t_avg)
        print(" Interpolating E_V...")
        E_base=zeros(len(t_base))
        if ti>0:
            E_base[:ti]=nan
        if tf<len(t_base)-1:
            E_base[tf:]=nan
        E_base[ti:tf]=interpolate(t_avg,E_avg,t_base[ti:tf],margin)
        print("  done!")

        return E_base

    def get_ELF(orbit,t_base,margin=30.,subset=False):
        dsp=pycdf.CDF(basedir+orbit+'/dsp_v58_'+orbit+'.cdf')
        print("Getting Time array...")
        t_elf=dsp['TIME']

        if subset:
            ti=0
            tf=len(t_base)-1
            i=0
            while(t_elf[i]<t_base[0]-margin):
                t_min=i
                i+=1
            print(" found t_min...")
            while(t_elf[i]<t_base[-1]+margin):
                t_max=i
                i+=1
            print(" found t_max...")
        else:
            t_min=0
            t_max=len(t_elf)
            i=0
            while(t_base[i]<t_elf[0]):
                ti=i+1
                i+=1
            print(" found ti...")
            while(i<len(t_base) and (t_base[i]<t_elf[-1])):
                tf=i
                i+=1
            print(" found tf...")
        tnew=zeros(t_max-t_min)
        for j in range(len(tnew)):
            tnew[j]=t_elf[t_min+j]
        t_elf=tnew
        print("  done!")

        print("Getting frequencies...")
        f=zeros(512)
        for i in range(512):
            f[i]=dsp['FREQ'][i]
        print("  done!")

        print("Getting DSP data...")
        spec=zeros((len(t_elf),512))
        for j in range(len(t_elf)):
            spec[j]=dsp['DSP'][j]
        print(" Integrating DSP data...")
        P=zeros(len(t_elf))
        for i in range(len(t_elf)):
            d=array(spec[i])
            for j in range(511):
                P[i]+=10**mean(d[j:j+2])*(f[j+1]-f[j])*1000.
        print(" Smoothing DSP Power...")
        Ps=zeros(len(P))
        for i in range(len(t_elf)):
            if i<2:
                Ps[i]=mean(P[:i+3])
            elif i>len(Ps)-3:
                Ps[i]=mean(P[i-2:])
            else:
                Ps[i]=mean(P[i-2:i+3])
        print(" Interpolating DSP Power...")
        P_base=zeros(len(t_base))
        if ti>0:
            P_base[:ti]=nan
        if tf<len(t_base)-1:
            P_base[tf:]=nan
        P_base[ti:tf]=interpolate(t_elf,Ps,t_base[ti:tf],margin)
        print("  done!")
```

```
    return P_base


def get_B(orbit,t_base,margin=30.,subset=False):
    dcmag=pycdf.CDF(basedir+orbit+'/dcmag_'+orbit+'.cdf')
    print("Getting Time array...")
    t_B=dcmag['TIME']

    if subset:
        ti=0
        tf=len(t_base)-1
        i=0
        while(t_B[i]<t_base[0]-margin):
            t_min=i
            i+=1
        print(" found t_min...")
        while(t_B[i]<t_base[-1]+margin):
            t_max=i
            i+=1
        print(" found t_max...")
    else:
        t_min=0
        t_max=len(t_B)
        i=0
        while(t_base[i]<t_B[0]):
            ti=i+1
            i+=1
        print(" found ti...")
        while(i<len(t_base) and (t_base[i]<t_B[-1])):
            tf=i
            i+=1
        print(" found tf...")
    tnew=zeros(t_max-t_min)
    for j in range(len(tnew)):
        tnew[j]=t_B[t_min+j]
    t_B=tnew
    print("  done!")

    print("Getting dB_o array...")
    dBo=zeros(len(t_B))
    for j in range(len(dBo)):
        dBo[j]=dcmag['DB_FAC_O'][t_min+j]
    t_avg=arange(t_B[0],t_B[-1]+4.,4.)
    print(" Smoothing dB_o...")
    dBo_avg=smooth(dBo,t_B,t_avg)
    print(" Interpolating dB_o...")
    dBo_base=zeros(len(t_base))
    if ti>0:
        dBo_base[:ti]=nan
    if tf<len(t_base)-1:
        dBo_base[tf:]=nan
    dBo_base[ti:tf]=interpolate(t_avg,dBo_avg,t_base[ti:tf],margin)
    print("  done!")

    print("Getting dB_e array...")
    dBe=zeros(len(t_B))
    for j in range(len(dBe)):
        dBe[j]=dcmag['DB_FAC_E'][t_min+j]
    print(" Smoothing dB_e...")
    dBe_avg=smooth(dBe,t_B,t_avg)
    print(" Interpolating dB_e...")
    dBe_base=zeros(len(t_base))
    if ti>0:
        dBe_base[:ti]=nan
    if tf<len(t_base)-1:
        dBe_base[tf:]=nan
    dBe_base[ti:tf]=interpolate(t_avg,dBe_avg,t_base[ti:tf],margin)
    print("  done!")

    print("Getting dB_b array...")
    dBb=zeros(len(t_B))
    for j in range(len(dBb)):
        dBb[j]=dcmag['DB_FAC_B'][t_min+j]
    print(" Smoothing dB_b...")
    dBb_avg=smooth(dBb,t_B,t_avg)
    print(" Interpolating dB_b...")
    dBb_base=zeros(len(t_base))
    if ti>0:
        dBb_base[:ti]=nan
    if tf<len(t_base)-1:
        dBb_base[tf:]=nan
    dBb_base[ti:tf]=interpolate(t_avg,dBb_avg,t_base[ti:tf],margin)
    print("  done!")

    return dBo_base,dBe_base,dBb_base

def get_EBS(orbit,t_base,margin=30.):
    tic=time()
    print("Processing E-fields...")
    Ev=get_E(orbit,t_base,margin)
    toc=time()
    tet=toc-tic
    print(" ET: {0}".format(toc-tic))
    tic=time()
    print("Processing B-fields...")
    dBo,dBe,dBb=get_B(orbit,t_base,margin)
    toc=time()
    tet+=toc-tic
    print(" ET: {0}".format(toc-tic))
    tic=time()
    print("Calculating S...")
    S=Ev*(dBe*1e-9)/(4e-7*pi)
    print("  done!")
    toc=time()
    print(" ET: {0}".format(toc-tic))
    tet+=toc-tic

    print("\nTotal ET: {0}".format(tet))

    return Ev,dBo,dBe,dBb,S


def get_flux(I,E,dE,theta):
    Na,Nb=shape(I)
    dOmega=zeros((Na,Nb))

    for a in range(Na-1):
        for b in range(Nb):
            th1=theta[a,b]*pi/180.
            th2=theta[a+1,b]*pi/180.

            if abs(th1-th2)>pi:
                dth=abs(th1-th2)-2*pi
            else:
                dth=abs(th1-th2)

            dOmega[a,b]=pi*abs(sin(th1))*cos(th1)*dth

    int_wz=sum(I*dOmega,0)
    Jz=sum(dE/E*int_wz)
    return Jz

def get_flux2(I,E,dE,theta,KE,alpha):
    Na,Nb=shape(I)
    Jz=0

    for a in range(Na-1):
        for b in range(Nb):
            th1=theta[a,b]*pi/180.
            th2=theta[a+1,b]*pi/180.

            if abs(th1-th2)>pi:
                dth=abs(th1-th2)-2*pi
            else:
                dth=abs(th1-th2)

            dOmega=pi*abs(sin(th1))*cos(th1)*dth

            if type(E)==int:
                Jz+=I[a,b]*dOmega*dE[b]
            else:
                E_eff=E[b]-KE*cos(theta[a,b]-alpha)
                Jz+=I[a,b]*dOmega*dE[b]/E_eff

    return Jz

def get_spectra(orbit,species):
    ies=pycdf.CDF(basedir+orbit+'/ies_'+orbit+'.cdf')
    t=array(ies['TIME'])
    th=arange(0,360,360./64)
    Es=zeros((len(t),48))-1
    PAs=zeros((len(t),64))-1

    for i in range(len(t)):
        print(" {0}%...\r".format(int(float(i+1)/len(t)*100))),
        stdout.flush()
        Efa=array(ies['Eflux'][i])
        Ea=array(ies['energy'][i])
        PAa=array(ies['angle'][i])
        if (Ea==standard_E[species+' energy']).all() and not isnan(Efa).any():
            Es[i,:]=sum(Efa,0)/64.
            for j in range(64):
                for k in range(48):
                    if PAa[j,k]<0:
                        PAa[j,k]+=360.
                    l=max(find(th<=PAa[j,k]))
                    PAs[i,l]+=Efa[j,k]

    return Es,PAs,th
```

```
def cutoff(orbit,maxE):
    ies=pycdf.CDF(basedir+orbit+'/ies_'+orbit+'.cdf')
    print("Calculating ion cutoff energy with maximum {0} eV...".format(maxE))
    print(" Getting Time array...")
    t=array(ies['TIME'])
    print("  done!")
    print(" Getting dayside cusp array...")
    ilt=array(ies['ILAT'])
    mlt=array(ies['MLT'])
    cusp=zeros(len(ilt))
    for i in range(len(ilt)):
        if ilt[i]>70. and ilt[i]<80. and abs(12-mlt[i])<6.:
            cusp[i]=1
    if sum(cusp)==0:
        for i in range(len(ilt)):
            if ilt[i]>65. and ilt[i]<85. and abs(12-mlt[i])<7.:
                cusp[i]=1
    print("  done!")

    print(" Getting Flux list...")
    Ef=ies['Eflux']
    print(" Getting Energy list...")
    E=ies['energy']
    print(" Getting Pitch Angle list...")
    PA=ies['angle']
    print("  done!")

    N=nan*zeros((len(t),47))    # Flux arrays by energy
    m=zeros(47)                 # Mean flux over orbit by energy
    c=zeros(47)                 # percent of outflow measurements by energy

    k=0
    K=len(find(cusp==1))
    print(" Getting {0} flux arrays by energy...".format(K))
    for i in find(cusp==1):
        k+=1
        print("\r {0}%...".format(int(float(k)/K*100))),
        stdout.flush()
        Efa=array(Ef[i])
        Ea=array(E[i])
        s=1
        if (Ea==standard_E['ion energy']).all():
            PAa=array(PA[i])
            dE=abs(Ea[:-1]-Ea[1:])
            for j in range(len(dE)-1):
                N[i,j]=get_flux(Efa[:,j+1:j+2],Ea[j+1:j+2],dE[j:j+1],
:::                                PAa[:,j+1:j+2])
    print("  done!")

    print(" Averaging fluxes...")
    for j in range(47):
        m[j]=nanmean(N[:,j])
        if sum(isnan(N[:,j]))==len(N[:,j]):
            c[j]=nan
        else:
            c[j]=float(sum(N[:,j]<0))/float(len(N[:,j])-sum(isnan(N[:,j])))
    print("  done!")

    k=46
    kmin=max(min(find(m/min(m)>1e-4)),min(find(Ea[1:]<maxE)))
    kmin=max(kmin,min(find(c>0.3)))
    found=False
    while not found:
        if (m[k]>0 and m[k-1]>0) or (k<=kmin):
            found=True
        else:
            k-=1
    cut=k+1

    return cut #,t,N,m,c

def get_EES(orbit,t_base,margin=30.,subset=False):
    ees=pycdf.CDF(basedir+orbit+'/ees_'+orbit+'.cdf')

    print("Getting Time array...")
    t_ees=ees['TIME']

    if subset:
        ti=0
        tf=len(t_base)-1
        i=0
        while(t_ees[i]<t_base[0]-margin):
            t_min=i
            i+=1
        print(" found t_min...")
        while(t_ees[i]<t_base[-1]+margin):
            t_max=i
            i+=1
        print(" found t_max...")
```

```
else:
    t_min=0
    t_max=len(t_ees)
    i=0
    while(t_base[i]<t_ees[0]):
        ti=i+1
        i+=1
    print(" found ti...")
    while(i<len(t_base) and (t_base[i]<t_ees[-1])):
        tf=i
        i+=1
    print(" found tf...")
tnew=zeros(t_max-t_min)
for j in range(len(tnew)):
    tnew[j]=t_ees[t_min+j]
t_ees=tnew
print("  done!")

print("Getting Flux list...")
Ef=ees['Eflux'][t_min:t_max]
print("  done!")
print("Getting Energy array...")
E=ees['energy'][t_min:t_max]
print("  done!")
print("Getting Pitch Angle list...")
PA=ees['angle'][t_min:t_max]
print("  done!")

print("Getting flux arrays...")
Jz=zeros(len(t_ees))
NJz=zeros(len(t_ees))

for i in range(len(t_ees)):
    print("\r {0}%...".format(int(float(i+1)/len(t_ees)*100))),
    stdout.flush()
    Ea=array(E[t_min+i])
    if not isnan(Ea).any():
        Efa=array(Ef[t_min+i])
        PAa=array(PA[t_min+i])
        dE=abs(Ea[:-1]-Ea[1:])
        Eh=0
        El=min(find(Ea<50.))
        if len(Ea[Eh:El])==0:
            NJz[i]=nan
            Jz[i]=nan
```

```
                else:
                    NJz[i]=get_flux(Efa[:,Eh:El],Ea[Eh:El],dE[Eh:El],PAa[:,Eh:El])
                    Jz[i]=get_flux(Efa[:,Eh:El],1,dE[Eh:El],PAa[:,Eh:El])
            else:
                NJz[i]=nan
                Jz[i]=nan
#    print("\n Smoothing flux arrays...")
#    t_avg=arange(t_ees[0],t_ees[-1]+4.,4.)
#    Jz_avg=smooth(Jz,t_ees,t_avg)
#    NJz_avg=smooth(NJz,t_ees,t_avg)

    print(" Interpolating flux arrays...")
    Jz_base=zeros(len(t_base))
    NJz_base=zeros(len(t_base))
    if ti>0:
        Jz_base[:ti]=nan
        NJz_base[:ti]=nan
    if tf<len(t_base)-1:
        Jz_base[tf:]=nan
        NJz_base[tf:]=nan
    Jz_base[ti:tf]=interpolate(t_ees,Jz,t_base[ti:tf],margin)
    NJz_base[ti:tf]=interpolate(t_ees,NJz,t_base[ti:tf],margin)
    print("  done!")

    return Jz_base,NJz_base

def get_IES(orbit,t_base,margin=30.,subset=False):
    ies=pycdf.CDF(basedir+orbit+'/ies_'+orbit+'.cdf')

    print("Getting Time array...")
    t_ies=ies['TIME']

    if subset:
        i=0
        while(t_ies[i]<t_base[0]-margin):
            t_min=i
            i+=1
        print(" found t_min...")
        while(t_ies[i]<t_base[-1]+margin):
            t_max=i
            i+=1
        print(" found t_max...")
    else:
        t_min=0
        t_max=len(t_ies)
        i=0
        while(t_base[i]<t_ies[0]):
            ti=i+1
            i+=1
        print(" found ti...")
        while(i<len(t_base) and (t_base[i]<t_ies[-1])):
            tf=i
            i+=1
        print(" found tf...")
    tnew=zeros(t_max-t_min)
    for j in range(len(tnew)):
        tnew[j]=t_ies[t_min+j]
    t_ies=tnew
    print("  done!")

    print("Getting Flux list...")
    Ef=ies['Eflux'][t_min:t_max]
    print("  done!")
    print("Getting Energy list...")
    E=ies['energy'][t_min:t_max]
    print("  done!")
    print("Getting Pitch Angle list...")
    PA=ies['angle'][t_min:t_max]
    print("  done!")

    Eh=cutoff(orbit,500.)

    print("Getting flux arrays...")
    Jz=zeros(len(t_ies))
    NJz=zeros(len(t_ies))

    for i in range(len(t_ies)):
        print("\r {0}%...".format(int(float(i+1)/len(t_ies)*100))),
        stdout.flush()
        Ea=array(E[t_min+i])
        if not isnan(Ea).any():
            Efa=array(Ef[t_min+i])
            PAa=array(PA[t_min+i])
            dE=abs(Ea[:-1]-Ea[1:])
            El=max(find(Ea>4.))+1
            if len(Ea[Eh:El])==0:
                NJz[i]=nan
                Jz[i]=nan
            else:
                NJz[i]=get_flux(Efa[:,Eh:El],Ea[Eh:El],dE[Eh-1:El-1],
:::                                PAa[:,Eh:El])
                Jz[i]=get_flux(Efa[:,Eh:El],1,dE[Eh-1:El-1],PAa[:,Eh:El])
        else:
            NJz[i]=nan
            Jz[i]=nan
#    print("\n Smoothing flux arrays...")
#    t_avg=arange(t_ies[0],t_ies[-1]+4.,4.)
#    Jz_avg=smooth(Jz,t_ies,t_avg)
#    NJz_avg=smooth(NJz,t_ies,t_avg)

    print("\n Interpolating flux arrays...")
    Jz_base=zeros(len(t_base))
    NJz_base=zeros(len(t_base))
    if ti>0:
        Jz_base[:ti]=nan
        NJz_base[:ti]=nan
    if tf<len(t_base)-1:
        Jz_base[tf:]=nan
        NJz_base[tf:]=nan
    Jz_base[ti:tf]=interpolate(t_ies,Jz,t_base[ti:tf],margin)
    NJz_base[ti:tf]=interpolate(t_ies,NJz,t_base[ti:tf],margin)
    print("  done!")

    return Jz_base,NJz_base,[Ea[Eh],Ea[El-1]]

def get_IES2(orbit,t_base,margin=30.,subset=False):
    ies=pycdf.CDF(basedir+orbit+'/ies_'+orbit+'.cdf')
    orb=pycdf.CDF(basedir+orbit+'/orbit_'+orbit+'.cdf')

    print("Getting Time array...")
    t_ies=ies['TIME']

    if subset:
        i=0
        while(t_ies[i]<t_base[0]-margin):
            t_min=i
            i+=1
        print(" found t_min...")
        while(t_ies[i]<t_base[-1]+margin):
            t_max=i
            i+=1
        print(" found t_max...")
    else:
        t_min=0
        t_max=len(t_ies)
        i=0
        while(t_base[i]<t_ies[0]):
            ti=i+1
            i+=1
        print(" found ti...")
        while(i<len(t_base) and (t_base[i]<t_ies[-1])):
            tf=i
            i+=1
        print(" found tf...")
    tnew=zeros(t_max-t_min)
    for j in range(len(tnew)):
        tnew[j]=t_ies[t_min+j]
    t_ies=tnew
    print("  done!")

    print("Getting Flux list...")
    Ef=ies['Eflux'][t_min:t_max]
    print("  done!")
    print("Getting Energy list...")
    E=ies['energy'][t_min:t_max]
    print("  done!")
    print("Getting Pitch Angle list...")
    PA=ies['angle'][t_min:t_max]
    print("  done!")

    print("Getting FAST velocity...")
    Vo=array(orb['FA_VEL'])
    print(" vx...")
    vx=interpolate(t_base,Vo[:,0],t_ies,margin)
    print(" vy...")
    vy=interpolate(t_base,Vo[:,1],t_ies,margin)
    print(" vz...")
    vz=interpolate(t_base,Vo[:,2],t_ies,margin)
    print("  done!")

    print("Getting B-field direction...")
    Bo=array(orb['B_MODEL'])
    print(" Bx...")
    Bx=interpolate(t_base,Bo[:,0],t_ies,margin)
    print(" By...")
    By=interpolate(t_base,Bo[:,1],t_ies,margin)
    print(" Bz...")
    Bz=interpolate(t_base,Bo[:,2],t_ies,margin)
    print("  done!")
```

```
    print("Getting energy and correction angle...")
    KE=zeros(len(t_ies))
    alpha=zeros(len(t_ies))
    for i in range(len(t_ies)):
        vdB=vx[i]*Bx[i]+vy[i]*By[i]+vz[i]*Bz[i]
        v=sqrt(vx[i]**2+vy[i]**2+vz[i]**2)
        B=sqrt(Bx[i]**2+By[i]**2+Bz[i]**2)
        KE[i]=0.083575035*v**2      # 1/2 mv^2 for oxygen
        alpha[i]=arccos(vdB/v/B)
    print("  done!")

    print("Getting cutoff energy...")
    Eh=cutoff(orbit,500.)
    print("  done!")

    print("Getting flux arrays...")
    Jz=zeros(len(t_ies))
    NJz=zeros(len(t_ies))

    for i in range(len(t_ies)):
        print("\r {0}%...".format(int(float(i+1)/len(t_ies)*100))),
        stdout.flush()
        Ea=array(E[t_min+i])
        if not isnan(Ea).any():
            Efa=array(Ef[t_min+i])
            PAa=array(PA[t_min+i])
            dE=abs(Ea[:-1]-Ea[1:])
            El=max(find(Ea>4.))+1
            if len(Ea[Eh:El])==0:
                NJz[i]=nan
                Jz[i]=nan
            else:
                NJz[i]=get_flux2(Efa[:,Eh:El],Ea[Eh:El],dE[Eh-1:El-1],
:::                                 PAa[:,Eh:El],KE[i],alpha[i])
                Jz[i]=get_flux2(Efa[:,Eh:El],1,dE[Eh-1:El-1],PAa[:,Eh:El],
:::                                KE[i],alpha[i])
        else:
            NJz[i]=nan
            Jz[i]=nan

    print("\n Interpolating flux arrays...")
    Jz_base=zeros(len(t_base))
    NJz_base=zeros(len(t_base))
    if ti>0:
        Jz_base[:ti]=nan
```

```
            NJz_base[:ti]=nan
        if tf<len(t_base)-1:
            Jz_base[tf:]=nan
            NJz_base[tf:]=nan
        Jz_base[ti:tf]=interpolate(t_ies,Jz,t_base[ti:tf],margin)
        NJz_base[ti:tf]=interpolate(t_ies,NJz,t_base[ti:tf],margin)
        print("  done!")

        return Jz_base,NJz_base,[Ea[Eh],Ea[El-1]]

    def processFAST(orbits,getelf=True):
        tic=time()
        for o in orbits:
            print("Processing orbit {0} information...".format(o))
            orb=pycdf.CDF(basedir+o+'/orbit_'+o+'.cdf')
            print(" Extracting base time...")
            T_base=zeros(len(orb['TIME']))
            for j in range(len(T_base)):
                T_base[j]=orb['TIME'][j]
            print(" Extracting satellite position...")
            X=zeros((len(T_base),3))
            for j in range(len(T_base)):
                X[j]=orb['FA_POS'][j]
            print(" Extracting invariant latitude...")
            ILT=zeros(len(T_base))
            for j in range(len(T_base)):
                ILT[j]=orb['ILAT'][j]
            print(" Extracting magnetic local time...")
            MLT=zeros(len(T_base))
            for j in range(len(T_base)):
                MLT[j]=orb['MLT'][j]
            print(" Extracting altitude...")
            ALT=zeros(len(T_base))
            for j in range(len(T_base)):
                ALT[j]=orb['ALT'][j]
            print(" Creating in-cusp flag...")
            CUSPF=range(len(ILT))
            for j in range(len(CUSPF)):
                if (ILT[j]>70. and ILT[j]<80.):
                    CUSPF[j]=1
                else:
                    CUSPF[j]=0
            print("  done!")
            toc=time()
            print("Orbit {0} elapsed time: {1}".format(o,toc-tic))
            print("Processing orbit {0} DC fields...".format(o))
            Ev,dBo,dBe,dBb,S=get_EBS(o,T_base)
            toc=time()
            print("Orbit {0} elapsed time: {1}".format(o,toc-tic))

            if getelf:
                print("Processing orbit {0} AC fields...".format(o))
                elf=get_ELF(o,T_base)
                toc=time()
                print("Orbit {0} elapsed time: {1}".format(o,toc-tic))
            else:
                elf=[]

            print("Processing orbit {0} electrons...".format(o))
            eJz,eNJz = get_EES(o,T_base)
            toc=time()
            print("Orbit {0} elapsed time: {1}".format(o,toc-tic))

            print("Processing orbit {0} ions...".format(o))
            iJz,iNJz,E_ends = get_IES(o,T_base)
            toc=time()
            print("Orbit {0} elapsed time: {1}".format(o,toc-tic))

    #        print("Getting orbit {0} outflow times...".format(o))
    #        print(" Creating spectra...")
    #        Es,PAs,t,th=get_spectra(o,'ion')
    #        contourf(log10(PAs.T))
    #        ti=t[int(raw_input("Enter the start time for outflow: "))]
    #        tf=t[int(raw_input("Enter the stop time for outflow: "))]
    #        OUTFLOW=0*range(len(ILT))
    #        for j in range(len(OUTFLOW)):
    #            if T_base[j]>ti and T_base[j]<tf:
    #                OUTFLOW[j]=1
    #        print("  done!")

            print("Pickling orbit {0} data...".format(o))
            print(" Creating data structure...")
            data={}
            data['Time']=T_base
            data['Position']=X
            data['Invariant Position']=[ILT,MLT,ALT]
            data['In Cusp']=CUSPF
    #        data['Outflow']=OUTFLOW
            data['E along Vsc']=Ev
            data['dB fac']=[dBo,dBe,dBb]
            data['Poynting Flux']=S
            data['ELF Amplitude']=elf
            data['eln_Eflux']=eJz
            data['eln_Nflux']=eNJz
            data['ion_Eflux']=iJz
            data['ion_Nflux']=iNJz

            print(" Creating meta data structure...")
            units={}
            units['Time']='UNX sec'
            units['Position']='GEI km'
            units['Invariant Position']=['ILAT deg','MLT hrs','ALT km']
            units['In Cusp']='flag'
    #        units['Outflow']='flag'
            units['E along Vsc']='mV/m'
            units['dB fac']=['o nT','e nT','b nT']
            units['Poynting Flux']='mW/m**2'
            units['ELF Power']='(V/m)**2'
            units['eln_Eflux']='mW/m**2'
            units['eln_Nflux']='#/cm**2/s'
            units['ion_Eflux']='mW/m**2'
            units['ion_Nflux']='#/cm**2/s'

            meta={}
            meta['orbit']=o
            meta['time covered']=[T_base[0],T_base[-1]]
            meta['date processed']=gmtime(time())
            meta['software version']=softversion
            meta['python version']=version[:3]
            meta['units']=units
            meta['ion energies']=E_ends

            print(" Pickling data to file...")
            filename=basedir+o+"/FAST_"+o+"-"+softversion[8:]+"-py"+version[0]
    :::              +".pkl"
            f=open(filename,'wb')
            pickle.dump({'meta':meta,'data':data},f)
            f.close()
            print("  Orbit {0} processing complete.".format(o))

            toc=time()
            print("  Total elapsed time: {0}".format(toc-tic))
```

E.1.2 FAPlot.py

```
from matplotlib.pyplot import *
from pylab import r_,find
from numpy import sqrt,sin,cos,pi,array,zeros,nanmin,log10
import plot_setup
import pickle
import FAST
import calendar

basedir="/home/david/research/FAST/data/"
filesuff="-1.7-py2.pkl"

def update_pickle(orbits):
    for o in orbits:
        print("Orbit "+o)
        print(" Extracting data...")
        ies=FAST.pycdf.CDF(basedir+o+"/ies_"+o+".cdf")
        t=array(ies['TIME'])
        mlt=array(ies['MLT'])
        ilt=array(ies['ILAT'])

        f=open(basedir+o+"/FAST_"+o+filesuff,'rb')
        d=pickle.load(f)
        f.close()
        it=d['data']['Time']
        print("  done!")

        OUTFLOW=zeros(len(it))

        Es,PAs,th=FAST.get_spectra(o,'ion')
        figure()
        contourf(log10(PAs.T),N=75,antialiased=False)
        plot(mlt,'r-',linewidth=2)
        plot(ilt/2.,'m-',linewidth=2)

        ti=0
        while ti != -1:
            ti=input("Start time (-1 is done): ")
            tf=input("Stop time: ")
            if ti!= -1 and tf != -1:
                plot([ti,ti],[0,64])
                plot([tf,tf],[0,64])
                ok=raw_input("Good times?: ")
                if ok=='y':
                    for j in range(len(it)):
                        if it[j]>=t[ti] and it[j]<=t[tf]:
                            OUTFLOW[j]=1

            d['data']['Outflow']=OUTFLOW
            d['meta']['units']['Outflow']='flag'

            f=open(basedir+o+"/FAST_"+o+"-a"+filesuff,'wb')
            pickle.dump(d,f)
            f.close()

def gen_pole_plot(orbits=[],ofp=False):
    if not ofp:
        plot_setup.set_params(8,'square',14,linewidth=1.)
        p=plot([0,0],[-50,50],'k-')
        plot([-50,50],[0,0],'k-')
        plot([-40,40],[40,-40],'k-')
        plot([-40,40],[-40,40],'k-')
        x=r_[-10:10:1000j]
        plot(x,sqrt(100-x**2),'k-')
        plot(x,-sqrt(100-x**2),'k-')
        x=r_[-20:20:1000j]
        plot(x,sqrt(400-x**2),'k-')
        plot(x,-sqrt(400-x**2),'k-')
        x=r_[-30:30:1000j]
        plot(x,sqrt(900-x**2),'k-')
        plot(x,-sqrt(900-x**2),'k-')
        x=r_[-40:40:1000j]
        plot(x,sqrt(1600-x**2),'k-')
        plot(x,-sqrt(1600-x**2),'k-')
        axis([-68,68,-68,68])
        setp(gca(), 'xticklabels', [])
        setp(gca(), 'yticklabels', [])
        a=3*pi/8
        text(10*cos(a),10*sin(a),r'$80^\circ$')
        text(20*cos(a),20*sin(a),r'$70^\circ$')
        text(30*cos(a),30*sin(a),r'$60^\circ$')
        text(40*cos(a),40*sin(a),r'$50^\circ$')
        text(-7,51,'12 MLT')
        text(-67,-1.8,'18 MLT')
        text(51,-1.8,'06 MLT')
        text(-7,-54.3,'00 MLT')

    for orb in orbits:
        print("Orbit " + orb)
        f=open(basedir+orb+ "/FAST_" +orb+"-a"+filesuff, 'rb')
        d=pickle.load(f)
        f.close()
        ilt=d['data']['Invariant Position'][0]
        mlt=d['data']['Invariant Position'][1]
        of=d['data']['Outflow']

        i=[]
        m=[]
        o=[]

        for j in range(len(ilt)):
            if ilt[j]>=45.:
                i.append(90.-ilt[j])
                m.append((mlt[j]-6)*pi/12.)
                o.append(of[j])
        i=array(i)
        m=array(m)
        if not ofp:
            plot(i*cos(m),i*sin(m),'r-')

        io=[]
        mo=[]
        for j in range(len(i)):
            if o[j]==1:
                io.append(i[j])
                mo.append(m[j])
        io=array(io)
        mo=array(mo)
        if ofp:
            plot(io*cos(mo),io*sin(mo),'k-',linewidth=5.)

def gen_summary_plot(orbit):
    f=open(basedir+orbit+"/FAST_"+orbit+"-a"+filesuff,'rb')
    d=pickle.load(f)
    f.close()

    plot_setup.set_params(8.5,'long',12,1)
    start=raw_input("Enter start time as yyyy,m,d,h,m,s: ").split(',')
    for i in range(len(start)):
        start[i]=int(start[i])
    t0=calendar.timegm(start)
    t=(d['data']['Time']-t0)/60.
    figure()
```

```
subplot(711)
title("FAST Orbit "+orbit)
plot(t,d['data']['E along Vsc'],'k-')
oa=axis()
axis([0,17,oa[2],oa[3]])
plot([0,17],[0,0],'k--')
ylabel("E along V$_{sc}$\n[DC] (mV/m)\n",horizontalalignment='center')

subplot(712)
plot(t,d['data']['dB fac'][0],'r-')
plot(t,d['data']['dB fac'][1],'g-')
plot(t,d['data']['dB fac'][2],'b-')
legend(['o','e','b'])
oa=axis()
axis([0,17,oa[2],oa[3]])
plot([0,17],[0,0],'k--')
ylabel("$\delta$B fac\n[DC] (nT)\n",horizontalalignment='center')

subplot(713)
plot(t,d['data']['Poynting Flux'],'m-')
oa=axis()
axis([0,17,oa[2],oa[3]])
plot([0,17],[0,0],'k--')
ylabel("Poynting Flux\n[DC] (mW/m$^2$)\n",horizontalalignment='center')

subplot(714)
plot(t,d['data']['eln_Nflux'],'k-')
oa=axis()
axis([0,17,oa[2],oa[3]])
plot([0,17],[0,0],'k--')
ylabel("Electron Flux\n(#/cm$^2$/s)\n",horizontalalignment='center')

subplot(715)
plot(t,d['data']['eln_Eflux']*1.6022e-12,'r-')
oa=axis()
axis([0,17,oa[2],oa[3]])
plot([0,17],[0,0],'k--')
ylabel("Electron\nEnergy Flux\n(mW/m$^2$)\n",horizontalalignment='center')

subplot(716)
plot(t,sqrt(d['data']['ELF Amplitude']),'g-')
oa=axis()
axis([0,17,oa[2],oa[3]])
plot([0,17],[0,0],'k--')
ylabel("ELF Amplitude\n(V/m)\n",horizontalalignment='center')
```

```
    subplot(717)
    plot(t,-d['data']['ion_Nflux'],'b-')
    oa=axis()
    axis([0,17,oa[2],oa[3]])
    plot([0,17],[0,0],'k--')
    ylabel("Ion FLux\n(#/cm$^2$/s)\n",horizontalalignment='center')
    xlabel("Time from {0:04d}/{1:02d}/{2:02d} {3:02d}:{4:02d}:{5:02d} (min)"
:::         .format(start[0],start[1],start[2],start[3],start[4],start[5]))
```

E.1.3 FAST_stats.py

```
from matplotlib.pyplot import *
from pylab import find
from numpy import array,zeros,arange,isnan,nansum,nanmin,log10,r_,delete
from scipy.stats import stats
import pickle
import plot_setup

basedir="/home/david/research/FAST/data/"
filesuff="-a-1.71-py2.pkl"

def get_averages(orbits,has_elf=True,alt_scale=False,length='Full'):
    if length=='Full':
        iNf=zeros(len(orbits))
        S=zeros(len(orbits))
        eNf=zeros(len(orbits))
        eEf=zeros(len(orbits))
        elf=zeros(len(orbits))
        nep=zeros(len(orbits))
        orbit=orbits
    else:
        iNf=[]
        S=[]
        eNf=[]
        eEf=[]
        elf=[]
        nep=[]
        orbit=[]

    for i in range(len(orbits)):
        o=orbits[i]
        f=open(basedir+o+"/FAST_"+o+filesuff,'rb')
        d=pickle.load(f)
        f.close()
        infr=d['data']['ion_Nflux']
        ofr=d['data']['Outflow']
        sr=d['data']['Poynting Flux']
        enfr=d['data']['eln_Nflux']
        eefr=d['data']['eln_Eflux']
        if has_elf:
            elfr=d['data']['ELF Amplitude']

        if alt_scale:
            alt=d['data']['Invariant Position'][2]
```

```
                infr*=(1000./alt)**(-3./2)
                sr*=(1000./alt)**(-3./2)
                enfr*=(1000./alt)**(-3./2)
                eefr*=(1000./alt)**(-3./2)

            if length=='Full':
                iNf[i]=nansum(infr*ofr)/nansum(ofr)
                S[i]=nansum(sr*ofr)/nansum(ofr)
                eNf[i]=nansum(enfr*ofr)/nansum(ofr)
                eEf[i]=nansum(eefr*ofr)/nansum(ofr)
                if has_elf:
                    elf[i]=nansum(elfr*ofr)/nansum(ofr)
                nep[i]=nansum(get_nep(o,alt_scale)*ofr)/nansum(ofr)
            else:
                idx=find(d['data']['Outflow']==1)
                j=idx[0]
                while j<=idx[-1]:
                    iNf.append(nansum(d['data']['ion_Nflux'][j:j+length]
:::                             *d['data']['Outflow'][j:j+length])
:::                             /nansum(d['data']['Outflow'][j:j+length]))
                    S.append(nansum(d['data']['Poynting Flux'][j:j+length]
:::                             *d['data']['Outflow'][j:j+length])
:::                             /nansum(d['data']['Outflow'][j:j+length]))
                    eNf.append(nansum(d['data']['eln_Nflux'][j:j+length]
:::                             *d['data']['Outflow'][j:j+length])
:::                             /nansum(d['data']['Outflow'][j:j+length]))
                    eEf.append(nansum(d['data']['eln_Eflux'][j:j+length]
:::                             *d['data']['Outflow'][j:j+length])
:::                             /nansum(d['data']['Outflow'][j:j+length]))
                    if has_elf:
                        elf.append(nansum(d['data']['ELF Amplitude'][j:j+length]
:::                                 *d['data']['Outflow'][j:j+length])
:::                                 /nansum(d['data']['Outflow'][j:j+length]))
                    nep.append(nansum(get_nep(o)[j:j+length]
:::                             *d['data']['Outflow'][j:j+length])
:::                             /nansum(d['data']['Outflow'][j:j+length]))
                    orbit.append(o)
                    j+=length

        if not length=='Full':
            iNf=array(iNf)
            S=array(S)
            eNf=array(eNf)
            eEf=array(eEf)
            if has_elf:
```

```
            elf=array(elf)
        nep=array(nep)

    return iNf,S,eNf,eEf,elf,nep,orbit

def get_plot(x,y):
    remove=find(x<=0)
    x=delete(x,remove)
    y=delete(y,remove)
    remove=find(y<=0)
    x=delete(x,remove)
    y=delete(y,remove)
    remove=find(isnan(x))
    x=delete(x,remove)
    y=delete(y,remove)
    remove=find(isnan(y))
    x=delete(x,remove)
    y=delete(y,remove)

    plot_setup.set_params(8,'square',16,linewidth=3)
    slp,itc,r,p,stderr=stats.linregress(log10(x),log10(y))
    loglog(x,y,'ko')
    a=r_[axis()[0]:axis()[1]:100j]
    plot(a,a**slp*10**itc,'r-')
    text(min(x),max(y),"slope = {0:.3f}\nr = {1:.3f}\np = {2:.3e}"
:::      .format(slp,r,p))

def get_nep(orbit,alt_scale=False):
    f=open(basedir+orbit+"/FAST_"+orbit+filesuff,'rb')
    d=pickle.load(f)
    f.close()
    enf=d['data']['eln_Nflux']
    eef=d['data']['eln_Eflux']
    if alt_scale:
        alt=d['data']['Invariant Position'][2]
        enf*=(1000./alt)**(-3./2)
        eef*=(1000./alt)**(-3./2)
    return 2.134e-14*(enf**1.5/eef**0.5)

def get_4qplot(x,y):
    limx=nanmin(abs(x))/2.
    limy=nanmin(abs(y))/2.
    print("Using {0:.3e},{1:.3e}".format(limx,limy))
    plot(x,y,'k.')
    xscale('symlog',linthreshx=1e-3,linthreshy=1e5)
    yscale('symlog',linthreshx=1e-3,linthreshy=1e5)

def gen_4qp(orbits,xtype,ytype,full=False):
    for o in orbits:
        print o
        f=open(basedir+o+"/FAST_"+o+filesuff,'rb')
        d=pickle.load(f)
        f.close()
        if ytype=='ion_Nflux':
            c=-1
        else:
            c=1
        if xtype=='nep':
            X=get_nep(o)
            Y=d['data'][ytype]
        else:
            X=d['data'][xtype]
            Y=d['data'][ytype]
        valid=find(d['data']['Outflow']==1)
        if not isnan(X[valid]).all():
            if not full:
                scatter(X[valid],c*Y[valid],1)
            else:
                scatter(X,c*Y,1)

def count_orbits(orbits,T):
    n=zeros(len(T))
    for o in orbits:
        print o
        f=open(basedir+o+"/FAST_"+o+filesuff,'rb')
        d=pickle.load(f)
        f.close()
        of=d['data']['Outflow']
        t=d['data']['Time']
        if sum(of)>0:
            st=t[min(find(of==1))]
            sp=t[max(find(of==1))]
            i=0
            while i<len(T) and st>T[i]:
                i+=1
            j=i
            while j<len(T) and sp>T[i]:
                j+=1
            if i==j:
                n[i-1]+=1
            else:
                n[i-1]+=1
                n[j-1]+=1

    return n

def time_to_latitude(orbit,lat):
    f=open(basedir+orbit+"/FAST_"+orbit+filesuff,'rb')
    d=pickle.load(f)
    f.close()
    ilt=d['data']['Invariant Position'][0]
    alt=d['data']['Invariant Position'][2]
    of=d['data']['Outflow']
    inf=d['data']['ion_Nflux']*(1000./alt)**(-1.5)
    s=d['data']['Poynting Flux']*(1000./alt)**(-1.5)
    enf=d['data']['eln_Nflux']*(1000./alt)**(-1.5)
    eef=d['data']['eln_Eflux']*(1000./alt)**(-1.5)
    nep=get_nep(orbit,True)

    linf=zeros(len(lat))
    ls=zeros(len(lat))
    lenf=zeros(len(lat))
    leef=zeros(len(lat))
    lnep=zeros(len(lat))
    n=zeros(len(lat))

    for i in range(len(ilt)):

        if ilt[i]>lat[0] and of[i]==1:
            nogo=isnan(inf[i]) or isnan(s[i]) or isnan(enf[i]) or isnan(eef[i])
:::              or isnan(nep[i])
            if not nogo:
                j=0
                while j<len(lat) and lat[j]<ilt[i]:
                    j+=1
                j-=1
                linf[j]+=inf[i]
                ls[j]+=s[i]
                lenf[j]+=enf[i]
                leef[j]+=eef[i]
                lnep[j]+=nep[i]
                n[j]+=1
    return linf,ls,lenf,leef,lnep,n

def get_latavg(orbits,t,lat=arange(70.,80.,0.5)):
```

```
size=(len(t),len(lat))
inf=zeros(size)
s=zeros(size)
enf=zeros(size)
eef=zeros(size)
nep=zeros(size)
cts=zeros(size)

n=count_orbits(orbits,t)
used=[]
for i in range(len(t)):
    print("Time Range {0}".format(i))
    print(" Finding used orbits...")
    used.append([])
    for j in arange(n[i])+sum(n[:i]):
        used[i].append(orbits[int(j)])
    print(used[i])

    print("\n Getting values...")
    for o in used[i]:
        linf,ls,lenf,leef,lnep,lcts=time_to_latitude(o,lat)
        inf[i]+=linf
        s[i]+=ls
        enf[i]+=lenf
        eef[i]+=leef
        nep[i]+=lnep
        cts[i]+=lcts

return inf,s,enf,eef,nep,cts,used
```

E.2 CHAMP

E.2.1 champ.py

```
# champ.py:  tools for extracting and manipulating the datasets from the
#   CHAMP satellite.
#
#   Rev. beta-3       16 Nov 2011
#
# Beta status. Began adding routines to process champ data for study.


from sys import stdout
from numpy import nan,isnan,array,zeros,shape
from math import atan2,sqrt
from scipy.interpolate import interp1d
import cstrans2
import champformats
import pickle
import time
import calendar


# ::: not Py3 compatible; needs to be changed to input() for Py3.
# On import, prompts for dataset to work.  Calling the load() function
# allows for dataset changing without reimporting champ.py

version = 'b3'

filename=raw_input("Working dataset filename: ")
def load():
    global filename
    filename=raw_input("Working dataset filename: ")

def process_CHAMP_orb(directory,days):
    global filename
    for d in days:
        print(d)
        filename=directory+"CH-OG-3-RSO+CTS-CHA_"+d[:-1]+str(int(d[-1])-1)
:::              +"_22.dat"
        dat=extract_data()
        t=zeros(len(dat['dJ2']))
        T=[]
        Xgeo=[]
```

```
Vgeo=[]
for i in range(len(t)):
    t[i]=time_convert(dat['dJ2'][i],dat['sJ2'][i])
    if time.gmtime(t[i])[7]==int(d[5:]):
        T.append(t[i])
        Xgeo.append(dat['pos'][i])
        Vgeo.append(dat['vel'][i])

filename=directory+"CH-OG-3-RSO+CTS-CHA_"+d+"_10.dat"
dat=extract_data()
t=zeros(len(dat['dJ2']))
for i in range(len(t)):
    t[i]=time_convert(dat['dJ2'][i],dat['sJ2'][i])
    T.append(t[i])
    Xgeo.append(dat['pos'][i])
    Vgeo.append(dat['vel'][i])

filename=directory+"CH-OG-3-RSO+CTS-CHA_"+d+"_22.dat"
dat=extract_data()
t=zeros(len(dat['dJ2']))
for i in range(len(t)):
    t[i]=time_convert(dat['dJ2'][i],dat['sJ2'][i])
    if time.gmtime(t[i])[7]==int(d[5:]):
        T.append(t[i])
        Xgeo.append(dat['pos'][i])
        Vgeo.append(dat['vel'][i])

T=array(T)
Xgeo=array(Xgeo)*1e-6
Vgeo=array(Vgeo)*1e-10
print(shape(Xgeo))
ipos=zeros((len(T),3))
for i in range(len(T)):
    Xgei=cstrans2.geo2gei(Xgeo[i],T[i])
    Xmag=cstrans2.gei2mag(Xgei,T[i])
    ipos[i,0:2]=Xmag[1:3]
    ipos[i,2]=sqrt(sum(Xgeo[i]**2))

new_dat={}
new_dat['t']=T
new_dat['pos']=Xgeo
new_dat['vel']=Vgeo
new_dat['ipos']=ipos
fs=open(directory+'RSO_'+d+'-'+version+'.pkl','wb')
pickle.dump(new_dat,fs)
```

```
        fs.close()

def process_CHAMP_acc(directory,days):
    global filename
    for d in days:
        f="CH-OG-2-ACC+"+d+"_00.9.dat"
        filename=directory+f
        print(filename)
        dat=extract_data()
        t=zeros(len(dat['tim']))
        dcy=zeros(len(t))
        for i in range(len(t)):
            t[i]=calendar.timegm(dat['tim'][i])
            dcy[i]=(dat['acl'][i][1]-dat['lk'][0][1])*dat['lk'][1][1]
        dat['t']=t
        dat['dcy']=dcy
        fs=open(directory+'ACC_'+f[12:20]+'-'+version+'.pkl','wb')
        pickle.dump(dat,fs)
        fs.close()

# Routine checks to ensure working dataset matches the requested
# set type
def check_dataset(set_type):
    known_sets={'acc':'%chacc', 'pso':'DSIDP ', 'rso':'DSIDP '}

    if set_type not in known_sets:
        error("CD01", set_type)
        return 0

    dataset=open(filename,'r')
    format_id=dataset.read(6)
    if format_id != known_sets[set_type]:
        error("CD02", set_type)
        dataset.close()
        return 0
    else:
        print("Found correct file type " + format_id)
        dataset.close()
        return 1


# Data extractor, general routine.  Sets dataset type, checks the file
# for set consistency, and prompts to select data elements for extraction
def extract_data():
    extractor={'acc':acc_extract, 'pso':pso_extract, 'rso':pso_extract}
    # ::: not Py3 compatible; needs to be changed to input() for Py3.
    set_type=raw_input("Set type for dataset: ")
    print('')

    if set_type not in extractor:
        error("ED01")
        return []

    if not check_dataset(set_type):
        error("ED02")
        return []

    data = extractor[set_type]()
    return data



### data extraction routines ###
def pso_extract():
    line_format={'dJ2':float, 'sJ2':float, 'pos':float, 'vel':float,
:::             'orn':float, 'ngd':float, 'flg':str}
    data={}
    dataset=open(filename,'r')
    last='file unread'

    while(dataset.read(5) != 'ORBIT' and last != ''):
        last=dataset.readline()
    if(last == ''):
        error("EX01",'pso')
        return []
    last=dataset.readline()

    setkey={0:'dJ2', 1:'sJ2', 2:'pos', 3:'vel', 4:'orn', 5:'ngd', 6:'flg'}
    used=key_select(setkey)
    datasize={0:(1,6), 1:(1,11), 2:(3,12), 3:(3,12), 4:(3,7), 5:(1,5), 6:(4,1)}

    for key in used:
        data.update({key:[]})
    while(last != ''):
        for key in datasize.keys():
            if datasize[key][0]>1:
                new_data = []
                for i in range(datasize[key][0]):
                    new_data.append(line_format[setkey[key]](
:::                        dataset.read(datasize[key][1])))
            else:
                new_data = dataset.read(datasize[key][1])
                if new_data=='':
                    break
                else:
                    new_data=line_format[setkey[key]](new_data)
            if (setkey[key] in used and new_data != ''):
                data[setkey[key]].append(new_data)
        last=dataset.readline()

    dataset.close()
    return data


# accelerometer data extraction
def acc_extract():
    # define the format of the lines in the data file.  These formats
    # do not include the 3 byte label in the first three characters
    # of the line in the champ data files, which is accounted for in
    # the scanning for position before parsing the data.
    line_format={'tim':champformats.tim, 'acl':champformats.acl,
                 'aca':champformats.aca, 'att':champformats.att,
                 'thr':[], 'acc':[]}

    data={}           # initialize the extracted data set
    marked={}         # flag used to identify when a data type is missing
                      # from a given timestamp

    dataset=open(filename,'r')
    last='file unread'        # last is a place to dump the remainder of
                              # a read line that is preserved to identify
                              # the %eof

    while(dataset.read(10) != '+data_____' and last != ''):
        last=dataset.readline()     # Scan to find the data description
    if(last == ''):
        error("EX01",'acc')
        return []

    setkey={}         # If not at %eof, start extracting the data types
    i=0               # included in the file, 3 byte labels with spaces
    while(dataset.read(1) != '\n'):     # between them.
        setkey.update({i:dataset.read(3)})
        i += 1
```

```
used=key_select(setkey)     # run key selection routine with the keys
                            # found in the dataset.

if 'acl' in used:           # extract acl calibration constants
    while(dataset.read(10) != '+acl_k0___' and last != ''):
        last=dataset.readline()
    if(last == ''):
        error("EX02",'acl')
        return[]
    lk0=[0,0,0]
    for i in range(3):
        lk0[i]=float(dataset.read(16))
    lk0_app=int(dataset.read(2))
    last=dataset.readline()

    while(dataset.read(10) != '+acl_k1___' and last != ''):
        last=dataset.readline()
    if(last == ''):
        error("EX02",'acl')
        return []
    lk1=[0,0,0]
    for i in range(3):
        lk1[i]=float(dataset.read(16))
    lk1_app=int(dataset.read(2))
    last=dataset.readline()

    print("Linear ACC Calibration values:  "),
    print(lk0,lk1)
    data.update({'lk':[lk0,lk1]})

if 'aca' in used:           # extract aca calibration constants
    while(dataset.read(10) != '+aca_k0___' and last != ''):
        last=dataset.readline()
    if(last == ''):
        error("EX02", 'aca')
        return []
    ak0=[0,0,0]
    for i in range(3):
        ak0[i]=float(dataset.read(16))
    ak0_app=int(dataset.read(2))
    last=dataset.readline()

    while(dataset.read(10) != '+aca_k1___' and last != ''):
        last=dataset.readline()
    if(last == ''):
        error("EX02", 'aca')
    ak1=[0,0,0]
    for i in range(3):
        ak1[i]=float(dataset.read(16))
    ak1_app=int(dataset.read(2))
    last=dataset.readline()

    print("Angular ACC calibration values: "),
    print(ak0,ak1)
    data.update({'ak':[ak0,ak1]})

for key in used:    # load data and marked dictionaries with used keys
    data.update({key:[]})
    marked.update({key:1})  # flags set to 1 to prevent initial
                            # timestamp from triggering a line of nan's
                            # for the other extracted variables

key=dataset.read(3)     # scan to data lines
while(key != 'tim' and last != ''):
    last=dataset.readline()
    key=dataset.read(3)

while(key != ''):       # parse until %eof

    if (key == 'tim'):  # first fill in data holes of used types
        for i in used:  # with nan's for previous timestamp
            if marked[i] == 0:
                new_data=[]
                for j in range(len(line_format[i])):
                    if line_format[i][j][0] != 'blank':
                        new_data.append(nan)
                data[i].append(new_data)
            else:
                marked[i]=0 # reset flag

### Parsing routine
# currently set to avoid use of thr and acc until coded

    # check if key is used, otherwise pass to the next line
    if (key in used and key not in ('thr','acc')):

        new_data=[]
        for i in range(len(line_format[key])):
            # line definitions with blanks should be parsed across
            if line_format[key][i][0]=='blank':
                dataset.read(line_format[key][i][1]+1)

            # all others are formatted according to their datatype
            # as defined in champformats.py
            else:
                new_data.append({'int':int,'float':float,'bin':str
:::                 }[line_format[key][i][0]](
:::                 dataset.read(line_format[key][i][1]+1)))

        # add the line of data and flag marked as done
        data[key].append(new_data)
        marked[key]=1

    ### temporary warning until thr and acc code is done ###
    if (key in used and key in ('thr','acc')):
        print("Not programmed for {0} yet.".format(key))

    # final step to parse line is to finish the readline (ideally only
    # picks up the \n at the line end) and read the key on the next
    # line.  Returns to while loop.
    last=dataset.readline()
    key=dataset.read(3)

# Finished parsing, close the file and return the extracted data
### currently does not return calibration values, only displayed on
### screen.
dataset.close()
return data


# key_select() and key_menu() are used to modify the extracted data set
# selection visually.

def key_select(setkey):
choice = 'z'
used=setkey.keys()  # defaults to using all extracted keys
key_menu(setkey)
while (choice != 'c'):
    print("> Current set uses {0}  (order not preserved)".format(used))

    print("> Enter 'd' to redisplay, 'a' to add, 'r' to remove, "),
    print("or 'c' to continue.")
    choice=raw_input("> ")
    if (choice == 'd'):
        key_menu(setkey)
    if (choice in ['a','r']):
        print("> Enter a value to {0}: ".format({'a':'add', 'r':'remove'
:::                 }[choice]))
        entry = int(raw_input("> "))
        if entry in {'a':setkey.keys(),'r':used}[choice]:
            {'a':used.append,'r':used.remove}[choice](entry)
        else:
            print("No value with key {0}".format(entry))

used_names=[]
for i in used:
    used_names.append(setkey[i])
return used_names


def key_menu(menu):
print('')
for i in menu.keys():
    print(" : {0} : {1}:".format(i,menu[i]))
print('')




### Error Code Definitions for all functions ###

def error(code, *info):
if (len(info) == 0):
    info=['']
message = {
    "CD01" : "Unknown set type {0}".format(info[0]),
    "CD02" : "Incorrect file type for set type {0}".format(info[0]),
    "ED01" : "Unknown set type for extraction",
    "ED02" : "Data set not compatible with set type",
    "EX01" : "Extractor {0} found no data head".format(info[0]),
    "EX02" : "Extractor failed to find {0} calibration values"
::                  .format(info[0])
}
print("ERR {0}: ".format(code) + message[code])


### Data Analysis Routines ###
```

```
def time_convert(d,s):
    J2000=946728000.0
    return J2000+24.*360.*d+s/1e6

def interpolate(x,y,width=20):
    y_new=zeros(len(y))
    for i in range(len(y)):
        if isnan(y[i]):
            x2=[]
            y2=[]
            for j in range(i-width/2,i+width/2):
                if isnan(y[j])==False:
                    x2.append(x[j])
                    y2.append(y[j])
            f=interp1d(x2,y2,kind='cubic')
            y_new[i]=f(x[i])
        else:
            y_new[i]=y[i]

    return y_new
```

E.2.2 champformats.py

```
# champformats: data file formatting details
# Rev. A1       20 Jan 2011

tim = [ ('int',4),       # year
        ('int',2),       # month
        ('int',2),       # day
        ('int',2),       # hours
        ('int',2),       # minutes
        ('float',10)  ] # seconds

acl = [ ('blank',4),
        ('float',13),   # linear ax
        ('float',13),   # linear ay
        ('float',13),   # linear az
        ('int',5),       # samples nx
        ('int',5),       # samples ny
        ('int',5)     ] # samples nz

aca = [ ('blank',4),
        ('float',13),   # angular aphi
        ('float',13),   # angular atheta
        ('float',13),   # angular apsi
        ('int',5),       # samples nphi
        ('int',5),       # samples ntheta
        ('int',5)     ] # samples npsi

att = [ ('bin',4),       # star camera flags
        ('float',13),   # attitude quaternions
        ('float',13),   # q1, q2, q3: vector part
        ('float',13),
        ('float',13),   # q4: scalar part
        ('float',4)   ] # quaternion accuracy
```

E.2.3 CHAMP_stats.py

```
from matplotlib.pyplot import *
from pylab import find
from math import pi
from numpy import array,zeros,arange,isnan,nansum,nanmin,log10,r_,delete,fft,
:::                 nan,mean,concatenate
from scipy.stats import stats
from scipy.interpolate import interp1d
import process
import pickle
import plot_setup
import time

basedir="/home/david/research/CHAMP/data/"
filesuff="-b3.pkl"

def get_averages(orbits,bd_append,type='raw'):
    f_basedir="/home/david/research/FAST/data/"
    mt=[]
    mdcy=[]
    milt=[]
    mmlt=[]

    for o in orbits:
        print("Geting averages for orbit {0}".format(o))
        f=open(f_basedir+bd_append+o+"/FAST_"+o+"-a-1.71-py2.pkl",'rb')
        fd=pickle.load(f)
        f.close()
        i=min(find(fd['data']['Outflow']==1))
        j=max(find(fd['data']['Outflow']==1))+1
        doy=time.gmtime(fd['data']['Time'][i])[7]
        if time.gmtime(fd['data']['Time'][j])[7]!=doy:
            print("Error! Orbit {0} overlaps two days!".format(o))

        d1=time.gmtime(fd['data']['Time'][0])[7]
        d2=time.gmtime(fd['data']['Time'][-1])[7]
        if d1==d2:
            cd1={}
            f=open(basedir+"RSO_2002_{0:03}".format(d1)+filesuff,'rb')
            d=pickle.load(f)
            cd1['rso']=d
            f.close()
            f=open(basedir+"ACC_2002_{0:03}".format(d1)+filesuff,'rb')
            d=pickle.load(f)
            cd1[’acc’]=d
            f.close()

            cd2=cd1

        else:
            cd1={}
            f=open(basedir+"RSO_2002_{0:03}".format(d1)+filesuff,’rb’)
            d=pickle.load(f)
            cd1[’rso’]=d
            f.close()
            f=open(basedir+"ACC_2002_{0:03}".format(d1)+filesuff,’rb’)
            d=pickle.load(f)
            cd1[’acc’]=d
            f.close()

            cd2={}
            f=open(basedir+"RSO_2002_{0:03}".format(d2)+filesuff,’rb’)
            d=pickle.load(f)
            cd2[’rso’]=d
            f.close()
            f=open(basedir+"ACC_2002_{0:03}".format(d2)+filesuff,’rb’)
            d=pickle.load(f)
            cd2[’acc’]=d
            f.close()

        cd={}
        cd[’acc’]={}
        cd[’rso’]={}
        if d1==d2:
            cd[’acc’][’t’]=cd1[’acc’][’t’]
            cd[’acc’][’dcy’]=cd1[’acc’][’dcy’]
            cd[’acc’][’filtered t’]=cd1[’acc’][’filtered t’]
            cd[’acc’][’filtered day’]=cd1[’acc’][’filtered day’]
            cd[’rso’][’t’]=cd1[’rso’][’t’]
            cd[’rso’][’ipos’]=cd1[’rso’][’ipos’]
        else:
            cd[’acc’][’t’]=concatenate((cd1[’acc’][’t’],cd2[’acc’][’t’]))
            cd[’acc’][’dcy’]=concatenate((cd1[’acc’][’dcy’],cd2[’acc’][’dcy’]))
            if type==’change’:
                cd[’acc’][’filtered t’]=concatenate((
:::                 cd1[’acc’][’filtered t’],cd2[’acc’][’filtered t’]))
                cd[’acc’][’filtered day’]=concatenate((
:::                 cd1[’acc’][’filtered day’],cd2[’acc’][’filtered day’]))
            cd[’rso’][’t’]=concatenate((cd1[’rso’][’t’],cd2[’rso’][’t’]))
            cd['rso']['ipos']=concatenate((
:::                 cd1['rso']['ipos'],cd2['rso']['ipos']))

        t1=min(find(cd['rso']['t']>fd['data']['Time'][0]))
        t2=min(find(cd['rso']['t']>fd['data']['Time'][-1]))

        ilt1=fd['data']['Invariant Position'][0][i]
        ilt2=fd['data']['Invariant Position'][0][j]
        min_ilt=min(ilt1,ilt2)
        max_ilt=max(ilt1,ilt2)
        t=[]
        dcy=[]
        ILT=[]
        MLT=[]
        if type=='raw':
            f=interp1d(cd['acc']['t'],cd['acc']['dcy'],kind='linear')
        elif type=='change':
            f=interp1d(cd['acc']['filtered t'],cd['acc']['filtered day'],
:::                     kind='linear')

        for k in range(t1,t2):
            ilt=cd['rso']['ipos'][k,0]*180/pi
            mlt=cd['rso']['ipos'][k,1]
            if ilt > min_ilt and ilt < max_ilt:
                if mlt > 6. and mlt < 18.:
                    dcy.append(f(cd['rso']['t'][k]))
                    ILT.append(ilt)
                    MLT.append(mlt)
        t=mean(cd['rso']['t'][t1:t2])
        mt.append(t)
        mdcy.append(mean(dcy))
        milt.append(mean(ILT))
        mmlt.append(mean(MLT))

    return array(mt),array(mdcy),array(milt),array(mmlt)


def time_to_latitude(ends,days,lat):
    for d in days:
        print(d)
        f=open(basedir+"RSO_"+d+filesuff,'rb')
        rso=pickle.load(f)
        f.close()
        f=open(basedir+"ACC_"+d+filesuff,'rb')
        acc=pickle.load(f)
        f.close()
        tr=rso['t']
        ilt=rso['ipos'][:,0]*180./pi
        ta=acc['filtered t']
        if len(days)==1:
            a=min(find(ends[0]<ta))
            b=min(find(ends[1]<ta))
        else:
            if d==days[0]:
                a=min(find(ends[0]<ta))
                b=len(ta)
            else:
                a=0
                b=min(find(ends[1]<ta))
        print(a,b)
        ta=ta[a:b]
        ay=acc['filtered ay'][a:b]
        nilt=nan*zeros(len(ta))
        a=0
        b=len(ta)
        for i in range(len(ta)):
            if ta[i]<tr[0]:
                a=i+1
            if ta[i]>tr[-1]:
                b=i
        print(a,b)
        nilt[a:b]=process.interp(tr,ilt,ta[a:b],100.)

        lay=zeros(len(lat))
        n=zeros(len(lat))

        for i in range(len(ta)):

            if nilt[i]>lat[0] and nilt[i]<(2*lat[-1]-lat[-2]):
                nogo=isnan(ay[i])
                if not nogo:
                    j=0
                    while j<len(lat) and lat[j]<nilt[i]:
                        j+=1
                    j-=1
                    lay[j]+=ay[i]
                    n[j]+=1
    return lay,n

def get_latavg(t,lat=arange(70.,80.,0.5)):
    size=(len(t),len(lat))
    ay=zeros(size)
    cts=zeros(size)

    used=[]
    for i in range(len(t)):
        print("Time Range {0}".format(i))
        print(" Finding used sets...")
        used.append([])
        d0=time.gmtime(t[i])[7]
        if i<len(t)-1:
            d1=time.gmtime(t[i+1])[7]
            ends=[t[i],t[i+1]]
        else:
            d1=time.gmtime(2*t[i]-t[i-1])[7]
            ends=[t[i],2*t[i]-t[i-1]]

        day0="{0}_{1}".format(time.gmtime(t[i])[0],d0)
        day1="{0}_{1}".format(time.gmtime(t[i])[0],d1)
        if day0==day1:
            used[i].append(day0)
        else:
            used[i].append(day0)
            used[i].append(day1)

        print(" Allocating data...")
        ay[i],cts[i]=time_to_latitude(ends,used[i],lat)
    return ay,cts

def update_acc_pickle(days,notch=''):
    for d in days:
        print("Updating {0}...".format(d))
        f=open(basedir+"ACC_"+d+filesuff,'rb')
        acc=pickle.load(f)
        f.close()
        t=acc['t']
        ay=-acc['dcy']
        t_interp=arange(t[0],t[-1]+10.,10.)
        ay_interp=process.interp(t,ay,t_interp,100.)
        ay_fft=fft.fft(ay_interp)
        ay_frq=fft.fftfreq(ay_fft.size, d=10.)

        if notch=='':
            P=zeros(ay_fft.size)
            for i in range(P.size):
                P[i]=ay_fft[i].real**2+ay_fft[i].imag**2
            plot(ay_frq,P)
            notch = raw_input("max freq: ")

        for i in range(ay_fft.size):
            if abs(ay_frq[i])>notch:
                ay_fft[i]=0
        new_ay=fft.ifft(ay_fft).real
        acc['filtered ay']=new_ay
        acc['filtered t']=t_interp
        acc['filtered day']=ay_interp-new_ay
        f=open(basedir+"ACC_"+d+filesuff,'wb')
        pickle.dump(acc,f)
        f.close()

def create_pickle(days):
    for d in days:
        print("Creating Pickle for {0}...".format(d))
        f=open(basedir+"ACC_"+d+filesuff,'rb')
        acc=pickle.load(f)
        f.close()
        f=open(basedir+"RSO_"+d+filesuff,'rb')
        rso=pickle.load(f)
        f.close()
        t=arange(rso['t'][0]//10*10.+10.,rso['t'][-1]//10*10.+10.,10.)
        print(" interpolating invariant latitude...")
        ilt=process.interp(rso['t'],rso['ipos'][:,0]*180./pi,t,100.)
        print(" interpolating magnetic local time...")
        mlt=process.interp(rso['t'],rso['ipos'][:,1],t,100.)
        print(" interpolating altitude...")
        alt=process.interp(rso['t'],rso['ipos'][:,2],t,100.)
        print(" interpolating x velocity...")
        vx=process.interp(rso['t'],rso['vel'][:,0],t,100.)
        print(" interpolating y velocity...")
        vy=process.interp(rso['t'],rso['vel'][:,1],t,100.)
        print(" interpolating z velocity...")
        vz=process.interp(rso['t'],rso['vel'][:,2],t,100.)
        print(" interpolating decceleration...")
        dcy=process.interp(acc['t'],acc['dcy'],t,100.)

        data={'t':t, 'ipos':array([ilt,mlt,alt]), 'vel':array([vx,vy,vz]),
:::           'dcy':dcy}
        f=open(basedir+"CHAMP_"+d+filesuff,'wb')
        pickle.dump(data,f)
        f.close()
        print("  done!")


def find_upwelling(fa_orbit,fa_basedir):
    plot_setup.set_params(10,'default',16)
    f=open(fa_basedir+fa_orbit+'/FAST_'+fa_orbit+'-a-1.71-py2.pkl','rb')
    fd = pickle.load(f)
    f.close()

    d1=time.gmtime(fd['data']['Time'][0])[7]
    d2=time.gmtime(fd['data']['Time'][-1])[7]

    if d1==d2:
        f=open(basedir+'CHAMP_2002_{0:03}-b3.pkl'.format(d1),'rb')
        cd=pickle.load(f)
        f.close()
    else:
        f=open(basedir+'CHAMP_2002_{0:03}-b3.pkl'.format(d1),'rb')
        cd1=pickle.load(f)
        f.close()
        f=open(basedir+'CHAMP_2002_{0:03}-b3.pkl'.format(d2),'rb')
        cd2=pickle.load(f)
        f.close()
        cd={}
        cd['t']=concatenate((cd1['t'],cd2['t']))
        cd['ipos']=zeros((3,len(cd['t'])))
        cd['ipos'][0]=concatenate((cd1['ipos'][0],cd2['ipos'][0]))
        cd['ipos'][1]=concatenate((cd1['ipos'][1],cd2['ipos'][1]))
        cd['ipos'][2]=concatenate((cd1['ipos'][2],cd2['ipos'][2]))
        cd['vel']=concatenate((cd1['vel'],cd2['vel']))
        cd['dcy']=concatenate((cd1['dcy'],cd2['dcy']))

    i=min(find(fd['data']['Outflow']==1))
    j=max(find(fd['data']['Outflow']==1))
    ilt_min=min(fd['data']['Invariant Position'][0][i],
:::             fd['data']['Invariant Position'][0][j])
    ilt_max=max(fd['data']['Invariant Position'][0][i],
:::             fd['data']['Invariant Position'][0][j])

    cusp=zeros(len(cd['t']))
    fast=zeros(len(cd['t']))
    for k in range(cusp.size):
        if cd['ipos'][0][k]>ilt_min and cd['ipos'][0][k]<ilt_max:
            if cd['ipos'][1][k]>6. and cd['ipos'][1][k]<18.:
                cusp[k]=1
        if cd['t'][k]>fd['data']['Time'][i] and
:::             cd['t'][k]<fd['data']['Time'][j]:
            fast[k]=1

    t0=fd['data']['Time'][0]
    close()
    figure()
    subplot(212)
    plot(fd['data']['Time']-t0,fd['data']['Invariant Position'][0],'r-')
    plot(cd['t']-t0,cd['ipos'][0],'b-')
    axis([0,fd['data']['Time'][-1]-t0,-90,90])
    legend(['FAST','CHAMP'],loc='best')
    xlabel('Time (s)')
    ylabel('Invariant Latitude (deg)')
    subplot(211)
    plot(cd['t']-t0,-1000*cd['dcy'])
    fill_between(cd['t']-t0,0,2,where=(fast==1),facecolor='red',alpha=0.5)
    fill_between(cd['t']-t0,0,2,where=(cusp==1),facecolor='green',alpha=0.5)
    axis([0,fd['data']['Time'][-1]-t0,0,1.1*max(-1000*cd['dcy'])])
    ylabel('Decceleration ($\mu$m/s$^2$)')

    good='n'
    while(good!='y'):
        ax=gca()
        if len(ax.lines)>3:
            ax.lines=ax.lines[:3]
        draw()
        start=float(raw_input("Enter starting time: "))
        stop=float(raw_input("Enter stopping time: "))
        plot([start,start],[0,2],'k--')
        plot([stop,stop],[0,2],'k--')
        good=raw_input("Good times (y/n): ")

    a=min(find(cd['t']>start+t0))
    b=min(find(cd['t']>stop+t0))

    t_interp=concatenate((cd['t'][2*a-b:a],cd['t'][b:2*b-a]))
    dcy_interp=concatenate((cd['dcy'][2*a-b:a],cd['dcy'][b:2*b-a]))
    f=interp1d(t_interp,dcy_interp,kind='linear')

    base=zeros(b-a)
    for i in range(b-a):
        base[i]=f(cd['t'][a+i])

    plot(cd['t'][a:b]-t0,-1000*base,'r--')
    raw_input("Press Enter to continue.")
    spike=array(cd['dcy'][a:b]-base)
    subplot(212)
    ax=gca()
    ax.clear()
    plot(cd['t'][a:b]-t0,-1000*spike,'m-')
    plot(cd['t'][a:b]-t0,zeros(b-a),'k--')
    axis([cd['t'][a]-t0,cd['t'][b]-t0,min(-1000*spike)*0.9,max(-1000*spike)*1.1])
    return mean(-1000*spike),max(-1000*spike)
```

E.3 Coordinate System Transformation

E.3.1 cstrans2.py

```
# ::   cstrans.py   ::
#
#   Coordinate System Transformation tools
#       v0.2
#
#   David K. Olson      Jul 2011
#   NASA/Goddard Space Flight Center
#   david.olson@nasa.gov
#
#       Requires IGRF.py to do magnetic-aligned coordinate systems.
#
# ::::::::::::::::::::

from numpy import array,matrix,zeros,dot,cross,sign
from math import sqrt,sin,asin,cos,acos,tan,atan,atan2,pi
from datetime import datetime
from time import gmtime
from calendar import timegm
from spacepy import time as spt
import IGRF

known_systems=['GEO','GEI']

# centered dipole calculation from IGRF
# see eg. http://www.spenvis.oma.be/help/background/magfield/cd.html
def get_dipole(model,year,eccentric=False):
    g,h=IGRF.load_IGRF(model,year)
    B0=sqrt(g[1,0]**2+g[1,1]**2+h[1,1]**2)
    c11=sqrt(g[1,1]**2+h[1,1]**2)
    theta=acos(-g[1,0]/B0)
    phi=asin(-h[1,1]/c11)
    if eccentric:
        L0=2*g[1,0]*g[2,0]+sqrt(3)*(g[1,1]*g[2,1]+h[1,1]*h[2,1])
        L1=-g[1,1]*g[2,0]+sqrt(3)*(g[1,0]*g[2,1]+g[1,1]*g[2,2]+h[1,1]*h[2,2])
        L2=-h[1,1]*g[2,0]+sqrt(3)*(g[1,0]*h[2,1]-h[1,1]*g[2,2]+g[1,1]*h[2,2])
        E=(L0*g[1,0]+L1*g[1,1]+L2*h[1,1])/(4*B0**2)

        x=array([(L1-g[1,1]*E)/(3*B0**2),(L2-h[1,1]*E)/(3*B0**2),
:::                  (L0-g[1,0]*E)/(3*B0**2)])*6371.2
    else:
        x=array([cos(pi/2-theta)*cos(phi),cos(pi/2-theta)*sin(phi),
:::                  sin(pi/2-theta)])
    return theta,phi,x

def jd(t,Julian=False):
    t_ex=gmtime(t)
    y=t_ex[0]
    m=t_ex[1]
    d=t_ex[2]
    H=t_ex[3]
    M=t_ex[4]
    S=t_ex[5]+t-int(t)

    fd = d+(H+(M+(S/60.))/60.)/24.

    if m<=2:
        y-=1
        m+=12

    A=y//100
    if Julian:
        B=0
    else:
        B=2-A+A//4

    return int(365.25*(y+4716))+int(30.6001*(m+1))+fd+B-1524.5


# Greenwich mean Sidereal Time from Meeus
def gst(t):
    djm=jd(t)-2451545.0
    T=djm/36525.
    return (280.46061837 + 360.98564736629*djm+3.87933e-4*T**2-T**3/3.871e7)%360.
```

```
# Greenwich mean Sidereal Time from Russell
def gst2(t):
    date=gmtime(t)
    y=date[0]
    d=date[7]
    fd=(date[3]+(date[4]+(date[5]+(t-int(t)))/60.)/60.)/24.

    dj=365.*(y-1900)+(y-1901)/4.+d+fd-0.5
    T=dj/36525.
    return (279.690983+0.9856473354*dj+360.*fd+180.)%360.

def gst3(t):
    t=spt.Ticktock(t,'UNX')
    mjd=t.MJD[0]
    T0=(mjd-51544.5)/36525.0
    return (100.461+36000.770*T0+15.04107)

def sun(t):
    date=gmtime(t)
    y=date[0]
    d=date[7]
    fd=(date[3]+(date[4]+(date[5]+(t-int(t)))/60.)/60.)/24.

    dj=365.*(y-1900)+(y-1901)/4+d+fd-0.5
    T=dj/36525.
    vl=(279.696678+0.9856473354*dj)%360.
    gst=(279.690983+0.9856473354*dj+360.*fd+180.)%360.
    g=((358.475845+0.985600267*dj)%360.)*pi/180.
    slong=vl+(1.91946-0.004789*T)*sin(g)+0.020094*sin(2.*g)
    obliq=(23.45229-0.0130125*T)*pi/180.
    slp=(slong-0.005686)*pi/180.
    sind=sin(obliq)*sin(slp)
    cosd=sqrt(1.-sind**2)
    sdec=180./pi*atan(sind/cosd)
    srasn=180.-180./pi*atan2(sind/cosd/tan(obliq),-cos(slp)/cosd)

    r=149598000.
    x=r*cos(srasn*pi/180.)*cos(sdec*pi/180.)
    y=r*sin(srasn*pi/180.)*cos(sdec*pi/180.)
    z=r*sin(sdec*pi/180.)

    return array([x,y,z])
```

```
def geo2gei(x_geo,t):
    theta=gst2(t)*pi/180.
    x_gei=zeros(len(x_geo))
    x_gei[0]=x_geo[0]*cos(theta)-x_geo[1]*sin(theta)
    x_gei[1]=x_geo[0]*sin(theta)+x_geo[1]*cos(theta)
    x_gei[2]=x_geo[2]

    return x_gei

def gei2geo(x_gei,t):
    theta=gst2(t)*pi/180.
    x_geo=zeros(len(x_gei))
    x_geo[0]=x_gei[0]*cos(theta)+x_gei[1]*sin(theta)
    x_geo[1]=-1*x_gei[0]*sin(theta)+x_gei[1]*cos(theta)
    x_geo[2]=x_gei[2]

    return x_geo

def gei2gsm(x_gei,t):
    theta,phi,D=get_dipole(7,1995)
    Dp=geo2gei(D,t)
    S=sun(t)
    DpS=cross(Dp,S)
    Y=DpS/sqrt(sum(DpS**2))
    Z=cross(S,Y)

    x_gsm=zeros(3)
    x_gsm[0]=dot(S,x_gei)
    x_gsm[1]=dot(Y,x_gei)
    x_gsm[2]=dot(Z,x_gei)

    return x_gsm

def gei2sm(x_gei,t):
    theta,phi,D=get_dipole(7,1995)
    Dp=geo2gei(D,t)
    S=sun(t)
    DpS=cross(Dp,S)
    Y=DpS/sqrt(sum(DpS**2))
    X=cross(Y,Dp)

    x_sm=zeros(3)
    x_sm[0]=dot(X,x_gei)
    x_sm[1]=dot(Y,x_gei)
    x_sm[2]=dot(Dp,x_gei)
    return x_sm


def transform(x,fr,to):
    error=False
    if fr not in known_systems:
        print("Unknown system {0}.".format(fr))
        error=True
    if to not in known_systems:
        print("Unknown system {0}.".format(to))
        error=True
    if len(x) != 3:
        print("Vector length needs to be 3 components.")
        error=True

    if error==True:
        return []

    T={'geogei':1, 'geigeo':2}

    v=array([[x[0]],[x[1]],[x[2]]])
    return T[fr+to]*v

###
#def solar_pos_GEI(year,day,time):
#    if year<1901 or year>2099:
#        print("year out of range for solar position.")
#        return
#    else:
#        if len(time)==3:
#            ftime=(time[0].+(time[1]+(time[2]/60.))/60.)/24.
#        elif len(time)==1:
#            ftime=time/86400.
#        else:
#            print("time formatting error: use either [hh,mm,ss.ss] or ss.sss")
#            return
#
#        DJ=365*(year-1900)+(year-1901)/4+day+ftime-0.5
#        T=DJ/36525.
#        VL=(279.696678+0.9856473354*DJ)%360.
#        GST=(279.690983+0.9856473354*DJ+360*ftime+180.)%360.
#        G=(358.475845+0.985600267*DJ)%360.*180./pi
#        SLONG=VL+(1.91946-0.
###
```

```
def rotate_x(v,theta):
    R=array([[1.,0.,0.],[0.,cos(theta),-sin(theta)],[0.,sin(theta),cos(theta)]])
    return dot(R,v)

def rotate_y(v,theta):
    R=array([[cos(theta),0.,sin(theta)],[0.,1.,0.],[-sin(theta),0.,cos(theta)]])
    return dot(R,v)

def rotate_z(v,theta):
    R=array([[cos(theta),-sin(theta),0.],[sin(theta),cos(theta),0.],[0.,0.,1.]])
    return dot(R,v)

def gei2mag(Xgei,t):
    X=gei2geo(Xgei,t)
    theta,phi,off=get_dipole(7,1995,eccentric=True)
    Xm=rotate_y(rotate_z(X-off,-phi),-theta)

    S=gei2geo(sun(t),t)
    Sm=rotate_y(rotate_z(S-off,-phi),-theta)

    mrad=sqrt(sum(Xm**2))
    mlat=asin(Xm[2]/mrad)
    mlon=atan2(Xm[1],Xm[0])
    slon=atan2(Sm[1],Sm[0])

    ilat=sign(Xm[2])*acos(min(1.,sqrt(6371.2/mrad)*cos(mlat)))
    if slon-mlon>pi:
        alpha=slon-mlon-2*pi
    elif slon-mlon<-pi:
        alpha=slon-mlon+2*pi
    else:
        alpha=slon-mlon
    mlt=12.-alpha*12./pi

    return Xm,ilat,mlt
```

E.4 Other Code

E.4.1 nantools.py

```
from numpy import nansum
from math import sqrt

def nanmean(a):
    return nansum(a)/nansum(a/a)

def nanstd(a):
    return sqrt(nanmean((a-nanmean(a))**2))
```

E.4.2 plot_setup.py

```
from math import sqrt
from pylab import rcParams as rc

shapes={'default': 0.774193548387, 'golden mean': (sqrt(5)-1)/2,
:::     'square': 1., 'portrait': 11./8.5, 'landscape': 8.5/11.,
:::     'long': 1.5}
pti=72.27

def set_params(width='',shape='',fontsize='',linewidth=2.):
    if width=='':
        width=rc['figure.figsize'][0]
    if shape=='':
        w,h=rc['figure.figsize']
        mt=1-rc['figure.subplot.top']
        mb=rc['figure.subplot.bottom']
        ml=rc['figure.subplot.left']
        mr=1-rc['figure.subplot.right']
        factor=h*(1-mt-mb)/w/(1-ml-mr)
    else:
        if not shapes.has_key(shape):
            print("Unkown shape.")
            return
        factor=shapes[shape]
    if fontsize=='':
        fontsize=rc['text.fontsize']
    else:
        fontsize=int(fontsize)
    print("Using width {0}, factor {1}, fontsize {2}".format(
::          width,factor,fontsize))
    ml = 6.0225*fontsize/pti
    mr = 4.8180*fontsize/pti
    mt = 3.6135*fontsize/pti
    mb = 3.6135*fontsize/pti

    pw = width - ml - mr
    ph = factor*pw
    height = mt + ph + mb

    params = {  'lines.linewidth': linewidth,
                'figure.figsize': [width,height],
                'figure.subplot.bottom': mb/height,
                'figure.subplot.top': 1-mt/height,
                'figure.subplot.left': ml/width,
                'figure.subplot.right': 1-mr/width,
                'font.size': fontsize  }
    rc.update(params)
```

Bibliography

[1] E.N. Parker, Newton, Maxwell, and Magnetospheric Physics, in *Magnetospheric Current Systems, Geophys. Monogr. Ser.*, **118**, edited by S.-I. Ohtani et al., 1-10, AGU, Washington, D.C. (2000).

[2] McGraw-Hill Concise Encyclopedia of Science & Technology, *Troposphere*, (1984).

[3] M.C. Kelley, *The Earth's Ionosphere: Plasma Physics & Electrodynamics*, Elsevier Inc., London (2009).

[4] A.E. Hedin, *J. Geophys. Res.*, **96**, 1159 (1991).

[5] Data obtained via http://ccmc.gsfc.nasa.gov/modelweb/models/msis_vitmo.php (last retrieved 7 Feb 2012).

[6] C.Y. Johnson, "Ion and neutral composition of the ionosphere", *Annals of the IQSY*, **5**, 197 (1969).

[7] H. Alfvén, L. Danielsson, C.G. Fälthammer, and L. Lindberg, *Natural Electromagnetic Phenomena Below 30 kc/s*, pp. 33–48, edited by D.F. Bleil, Plenum, New York (1964).

[8] J. Larmor, *Rep. of the British Assoc.*, **87**, 159 (1919).

[9] IGRF details can be found at: http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html

[10] M.F. Smith and M. Lockwood, *Rev. Geophys.*, **34**, 233 (1996).

[11] N. Meyer-Vernet, *Basics of the Solar Wind, Cambridge Atmospheric and Space Science Series*, Cambridge University Press, Cambridge (2009).

[12] T.E. Moore and J.L. Horwitz, *Rev. Geophys.*, **45**, RG3002 (2007).

[13] M.F. Shea, R.D. Sharp, and M.B. McElroy, *J. Geophys. Res.*, **73**, 4199–4212 (1968).

[14] W.G. Pilipp, H. Miggenrieder, M.D. Montgomery, K.-H. Mühlhäuser, H. Rosenbauer, and R. Schwenn, *J. Geophys. Res.*, **92**, 1075–1092 (1987).

[15] G.R. Wilson, D.R. Weimer, J.O. Wise, and F.A. Marcos, *J. Geophys. Res.*, **111**, A10314 (2006).

[16] A.D. Richmond and J.P. Thayer, Ionospheric electrodynamics: A tutorial, in *Magnetospheric Current Systems, Geophys. Monogr. Ser.*, vol. **118**, edited by S.-I. Ohtani et al., 131-146, AGU, Washington, D.C. (2000).

[17] B. Gustavsson, M.T. Reitveld, N.V. Ivchenko, and M.J. Kosch, *J. Geophys. Res.*, **115**, A12332 (2010).

[18] E.N. Parker, *J. Geophys. Res.*, **62**, 509 (1957).

[19] L.G. Jacchia and J. Slowey, *J. Geophys. Res.*, **69**, 905-910 (1964).

[20] H. Lühr, M. Rother, W. Köhler, P. Ritter, and L. Grunwaldt, *Geophys. Res. Lett.*, **31** , 6805 (2004).

[21] J.H. Clemmons, J.H. Hecht, D.R. Salem, and D.J. Strickland, *Geophys. Res. Lett.*, **35**, L24103 (2008).

[22] K. Schlegel, H. Lühr, J.P. St. Maurice, G. Crowley, and C. Hackert, *Ann. Geophy.*, **23**, 1659 (2005).

[23] H.G. Demars and R.W. Schunk, *J. Atmos. Solar-Terr. Phy.*, **69**, 649 (2007).

[24] Various Authors, *Measurement Techniques in Space Plasmas: Particles, Geophys. Monogr. Ser.*, **102**, edited by R.F. Pfaff, J.E. Borovsky, and D.T. Young, AGU, Washington, D.C. (1998).

[25] J.L. Wiza, *Nucl. Inst. And Meth.*, **162**, 587-601 (1979).

[26] E.J. Gamboa, C.M. Huntington, E.C. Harding, and R.P. Drake, *Rev. Sci. Inst.*, **81**, 10E310 (2010).

[27] K. Furuya and Y. Hatano, *Int. J. of Mass Spectrom.*, **218**, 237 (2002).

[28] D.T. Young, *Measurement Techniques in Space Plasmas: Particles, Geophys. Monogr. Ser.*, **102**, 1 (1998).

[29] J. Oberheide, P. Wilhelms, and M. Zimmer, *Meas. Sci. Technol.*, **8**, 351 (1997).

[30] R.E. Ergun et al., *Geophys. Res. Lett.*, **25**, 2041 (1998).

[31] W.K. Peterson et al., *Geophys. Res. Lett*, **25**, 2081 (1998).

[32] R. Lysak, *Geophys. Res. Lett*, **25**, 2089 (1998).

[33] H. Lühr, S. Maus, and M. Rother, *J. Geophys. Res.*, **109**, A01306 (2004).

[34] H. Lühr, S. Maus, and M. Rother, *Gephys. Res. Lett.*, **29**, 1489 (2002).

[35] C. Stolle, H. Lühr, M. Rother, and G. Balasis, *J. Geophys. Res.*, **111**, A02304 (2006).

[36] R. Pfaff, C. Carlson, J. Watzin, D. Everett, and T. Gruner, *Space Sci. Rev.*, **98**, 1 (2001).

[37] R.E. Ergun et al., *Space Sci. Rev.*, **98**, 67 (2001).

[38] R.C. Elphic, J.D. Means, R.C. Snare, R.J. Strangeway, L. Kepko, and R.E. Ergun, *Space Sci. Rev.*, **98**, 151 (2001).

[39] C.W. Carlson, J.P. McFadden, P. Turin, D.W. Curtis, and A. Magoncelli, *Space Sci. Rev.*, **98**, 33 (2001).

[40] C. Reigber, H. Lühr, and P. Schwintzer, “Announcement of Opportunity for CHAMP”, CH-GFZ-AO-001, GFZ Potsdam, Germany (2001).

[41] S. Bruinsma, D. Tamagnan, and R. Biancale, *Planet Space Sci.*, **52**, 297 (2004).

[42] T.E. Moore *et al.*, *Geophys. Res. Lett.*, **26**, 2339 (1999).

[43] R.J. Strangeway et al., *J. Geophys. Res.*, **105**, 21,129 (2000).

[44] R.J. Strangeway, R.E. Ergun, Y.-J. Su, C.W. Carlson, and R.C. Elphic, *J. Geophys. Res.*, **110**, A03221 (2005).

[45] K.J. Zahnle and D.C. Catling, ”Our Planet’s Leaky Atmosphere,” *Scientific American*, May 2009.

[46] R.D. Sharp, R.G. Johnson, and E.G. Shelley, *J. Geophys. Res.*, **82**, 3324 (1977).

[47] D.J. Gorney, A. Clarke, D. Croley, J. Fennell, J. Luhmann, and P. Mizera, *J. Geophys. Res.*, **86**, 83 (1981).

[48] C.R. Chappell, T.E. Moore, and J.H. Waite Jr., *J. Geophys. Res.*, **92**, 5896 (1987).

[49] M.-C. Fok, T.E. Moore, and M.R. Collier, *J. Geophys. Res.*, **109**, A01206 (2004).

[50] R.L. Lysak and M. André, *Phys. Chem. Earth (C)*, **26**, 3 (2001).

[51] K.A. Lynch et al., *Ann. Geophys.*, **25**, 1967 (2007).

[52] M. Andre, *J. Atmos. Terr. Phys.*, **59**, 1687 (1997).

[53] Y. Zheng, T.E. Moore, F.S. Mozer, C.T. Russell, and R.J. Strangeway, *J. Geophys. Res.*, **110**, A07210 (2005).

[54] C.T. Russell, *Cosmic Electrodynamics*, **2**, 184 (1971).

[55] C.C. Finlay et al., *Geophys. J. Int.*, **183**, 1216 (2010).

[56] Available at http://realterm.sourceforge.net

[57] The Fast Auroral SnapshoT Explorer, http://sprgl.ssl.berkeley.edu/fast

[58] CDF Home Page, http://cdf.gsfc.nasa.gov

[59] The Information System and Data Center, http://isdc.gfx-potsdam.de

[60] Numerical Python, http://numpy.scipy.org

[61] Scientific Tools for Python, http://www.scipy.org

[62] The matplotlib Plotting Library for Python, http://matplotlib.sourceforge.net

[63] Python-based Tools for the Space Science Community, http://spacepy.lanl.gov

[64] C. Förste, P. Schwintzer, and C. Reigber, "The CHAMP Data Format", CH-GFZ-FD-001, GFZ Potsdam, Germany (2002).

[65] R. König, P. Schwintzer, and C. Reigber, "The CHAMP Orbit Format CHORB", CH-GFZ-FD-002, GFZ Potsdam, Germany (2001).

[66] P. Schwintzer, H. Lühr, C. Reigber, L. Grunwaldt, and C. Förste, "CHAMP Reference Systems, Transformations and Standards", CH-GFZ-RS-002, GFZ Potsdam, Germany (2002).